A COMPARATIVE ANALYSIS OF FILE ORGANIZATIONS

Barry Nicholas Bittner

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A COMPARATIVE ANALYSIS OF FILE ORGANIZATIONS

by

Barry Nicholas Bittner
and
Charles Lorain Carpenter, Jr.

Thesis Advisor: U. R. Kodres

June 1972

A Comparative Analysis of File Organizations

by

Barry Nicholas Bittner
Lieutenant Colonel, United States Marine Corps
B.S., United States Naval Academy, 1954

and

Charles Lorain Carpenter, Jr.
Major, United States Marine Corps
B.A., The University of Virginia, 1957

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the
NAVAL POSTGRADUATE SCHOOL
June, 1972

Thes
B54-11
[illegible]

ABSTRACT

Increasingly more sophisticated weaponry necessitates that U. S. military organizations insure timely and responsive tactical command and control systems. Automation is one obvious answer towards accomplishing this goal. This paper may be viewed as a simulation study of file organizations which are typical to command and control systems. It reports the findings of a comparative analysis of five different file organizations to determine their responsiveness to five types of commonly used application subroutines. It also uncovers areas for future research with respect to command and control systems' file organizations.

TABLE OF CONTENTS

I. INTRODUCTION

The potential enemies of the United States are developing or presently being equipped with increasingly more sophisticated weapons systems. Thus, future warfare promises to be more complex and faster-moving than ever before realized. This threat necessitates that the military organization of the United States insure that the efforts of all combat arms be closely coordinated and interleaved to achieve maximum combat effectiveness. Automation, one obvious answer to this goal, offers many promises for improved tactical effectiveness by providing faster response times, powerful computational aids, and more complete information conveniently available to enable decision-makers to better understand and coordinate the battlefield situation.

Over the years the Marine Corps has attempted to infuse automation into different levels of command. To a great extent, however, this has been an uncoordinated effort. In 1964 the Marine Corps initiated the development of an overall tactical command and control system now known as the Marine Tactical Command and Control System (MTACCS). See Figure 1. This system contains the following seven subsystems (1):

(1) Tactical Combat Operations System (TCO)

(2) Marine Air Command and Control System (MACCS)

(3) Marine Integrated Fire and Air Support System (MIFASS)

(4) Marine Integrated Personnel System (MIPS)

(5) Marine Integrated Logistics System (MILOGS)

(6) Marine Air Ground Intelligence System (MAGIS)

(7) Communications System (COMMS)

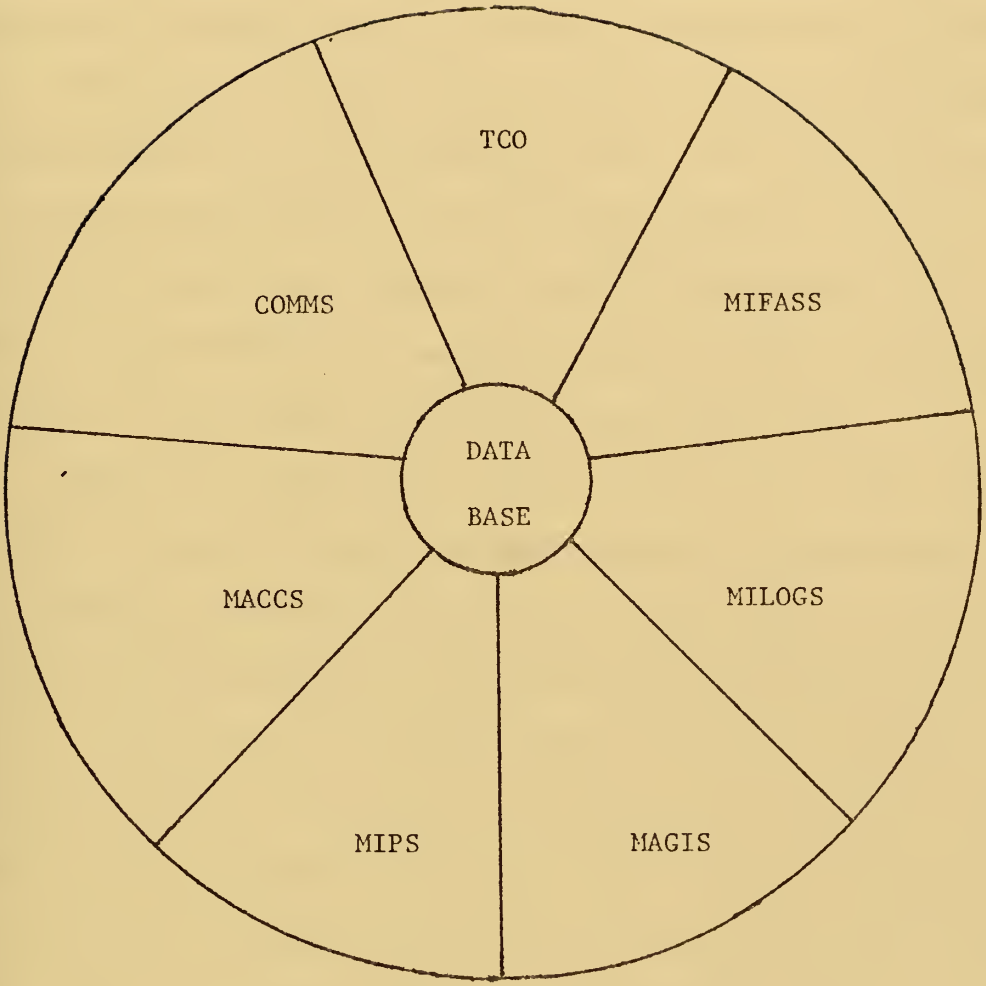


Figure 1.
Marine Tactical Command and Control System (MTACCS)

The Marine Integrated Fire and Air Support System (MIFASS) is currently undergoing a two year development and evaluation at the test bed of the Marine Corps Tactical Systems Support Activity at Camp Pendleton, California.

It has become obvious in the MIFASS development that with dynamically changing tactical situations, variable unit deployments, etc., the degree of change required of the date base will demand extreme flexibility in the handling of data. This type of flexibility in the field necessitates a reprogramming capability, but, in a battlefield environment such a solution would be unreasonable. Hence, some form of a generalized data management system (GDMS) would be required in order to free military units from this arduous task.

Without the sophisticated approach to software changes afforded by a GDMS, all message formats and user application programs have to be tied directly to fixed file organizations and formats. Each user application programmer must know precisely the location of every data field in the records so that this information can be accessed. As a result, different user application programs must be written to access the data fields in fixed file record format. This conventional approach is simple and straight-forward as long as the input formats, file formats and output formats never change. Inevitably user application program requirements change, resulting in a series of additional format changes to ensure compatability in all processing and program inputs and outputs. Insuring this compatability is not a trivial matter in that it will be costly in both time and human resources.

In order to avoid these types of problems, a GDMS can be used, making the maintenance and interaction with a data base a relatively

simple chore. Changes to files do not affect application programs or input formats. Conversely, changes to the input formats do not require reprogramming or file structure changes. In effect, a GDMS causes the data base to be independent of the user. This allows the tactical user to interact with the system with simplified procedures as he creates, deletes, or modifies data and/or message/display formats. By freeing the tactical user from lengthy and complicated data handling procedures, he is free to concentrate on his primary responsibility, that of reviewing, manipulating and reacting to the data content. (2)

To date, MIFASS contains seven application programs:

(1) Fire Mission Analysis

(2) Air Support Control

(3) Technical Fire Control

(4) Troop Safety

(5) Target Data

(6) Conflict Detection

(7) Mission Scheduling and Monitoring

Then programs are interactive. That is, each of the seven tactical application programs is dependent upon the other's outputs throughout various stages of processing and analysis. For example, prior to the completion of the Fire Mission Analysis application program, Troop Safety and Conflict Detection must interact in order to provide indications of unsafe conditions to the weapon selection display. This display is presented to a fire support coordinator who must make the final decision as to who will provide the fire power on the target to be attacked.

Data bases for military tactical command and control systems will be, out of necessity, quite large. It has been estimated from the results of load analysis that the memory requirements for MIFASS alone will be approximately 120 million bits.(3) The problem then lies in developing an effective file structure for ensuring responsiveness and efficiency to the demands placed upon it by command and control needs. For example, a substantial number of large files will be associated with the Marine Tactical Command and Control System. One of these files, for example, is the Decision Logic Table, which contains over 900 records.

The organization of a data base can be structured into any one on many configurations; there will be advantages and disadvantages to each. Quite naturally, it will be necessary to determine the primary application subroutines to be applied to the data base when accessing the file structure. In tactical systems, for example, responsiveness to queries must be considered paramount over other data base design criteria such as storage requirements or programming complexity. A review of the Fire Mission Analysis application program reveals that during its processing six application subroutine queries are used to access the data base. By those subroutine queries different lists of data are extracted from the data base upon which the program can then operate. For example, a query is made for the retrieval of the weapon list from the Decision Logic Table where all potentially acceptable weapons systems are listed. Subsequently a query is made for the retrieval of the units available with the proper weapons systems from the unit file. Thus, it can be concluded that one of the primary functions of the application subroutine queries will be to extract lists of data from the data base.

In a recent analysis of the Fire Mission Analysis application program's time processing profile, it was revealed that 43 per cent of the total execution time is consumed by searching the data base in response to application subroutine queries.(4) This emphasizes the importance of an efficient data base, one that will minimize the length of time that must be relegated to the searching and retrieving functions.

The primary purpose of the work done in this paper was to conduct a comparative analysis of five different file organizations and determine their responsiveness to five types of commonly used application subroutines associated with tactical command and control systems. A secondary purpose in the paper was to conduct exploratory research of file organizations. This area has been one that has not had adequate attention in the past and it was hoped to uncover areas for future research.

The remaining sections of this paper are organized in the following manner:

(1) Section II presents the definitions, file structures and search techniques used in the paper.

(2) Section III establishes the parameters of the application subroutines.

(3) Section IV outlines both the file structures and application processes used.

(4) Data gathered in the file organization comparison is presented in Section V.

(5) Section VI identifies possible future research in the area of file organization, specifically as it relates to the MTACCS test bed.

(6) Section VII outlines the conclusions found in the file organization comparison.

II. DATA BASE ORGANIZATION

A. DEFINITIONS

In this section a formal approach to the several file structures studied and their concomitant information retrieval schemes is presented. Each of these file structures is characterized and classified. Similiarly, the various methods of information retrieval utilized with these files are categorized.

Before examining the structure of any file or its retrieval schemes, it is necessary to define or otherwise establish a common reference to the principal terms employed in this paper. Figure 2 provides a model of a typical generalized file structure. All examples included with the definitions below make use of this figure. The tree in Figure 3 describes the hierarchy of the file structure. The definitions set forth below are in accordance with those presented by Hsiao and Harary.(5)

An ELEMENTARY DATA ITEM E is the smallest unit of information which is processed. For example, in Figure 2 each last name, rank and pointer is an elementary data item.

A RECORD R is an ordered collection of elementary data items. These elementary data items are the attributes which make up the record. Each attribute has a single value. In Figure 2 the values DOE, CPL, 0311 and 121 make up a record for the attributes: last name, rank, military occupational specialty (MOS) and pointer.

A KEYWORD K is any elementary data item within a record. It is the means by which a record is referenced. Keywords, may be subscripted K_i to indicate distinct values. In Figure 2 the last names Doe, Jones and Smith are keywords.

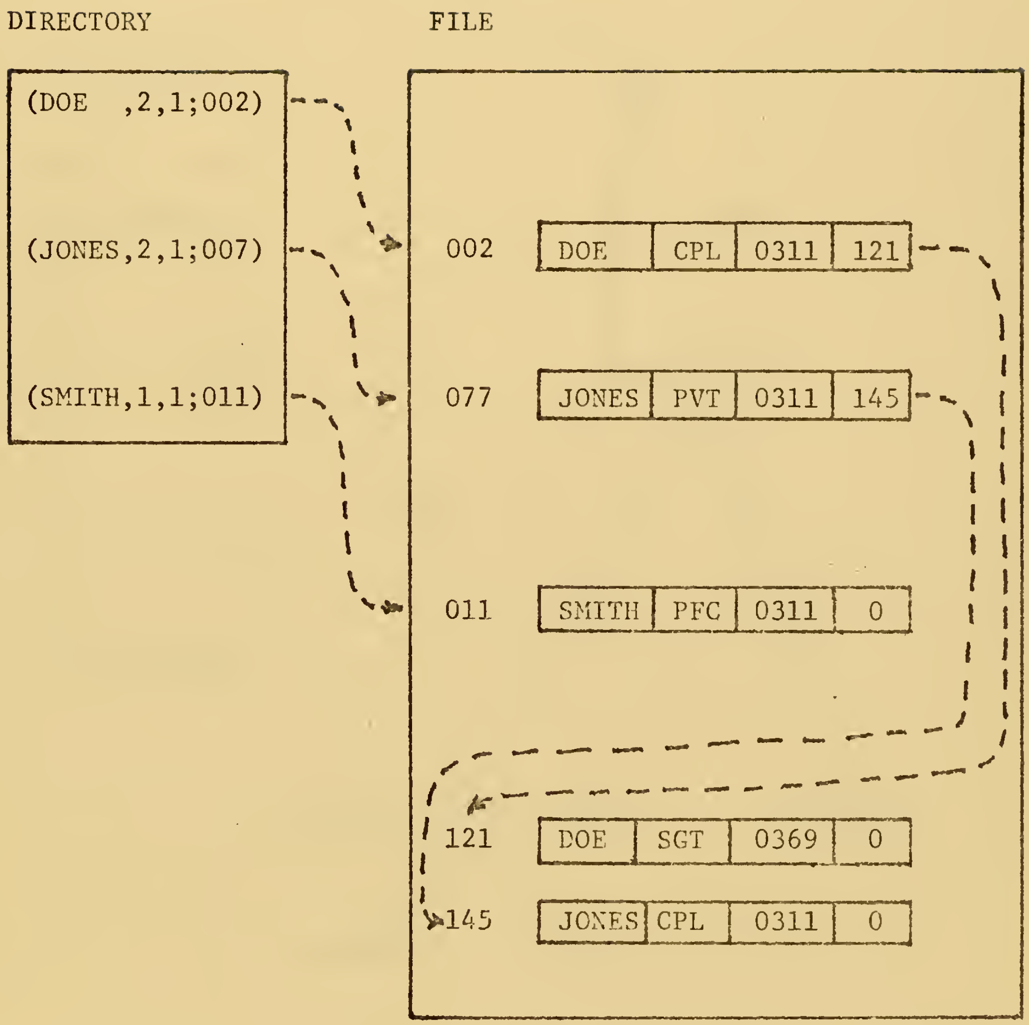


Figure 2.
Generalized File Structure

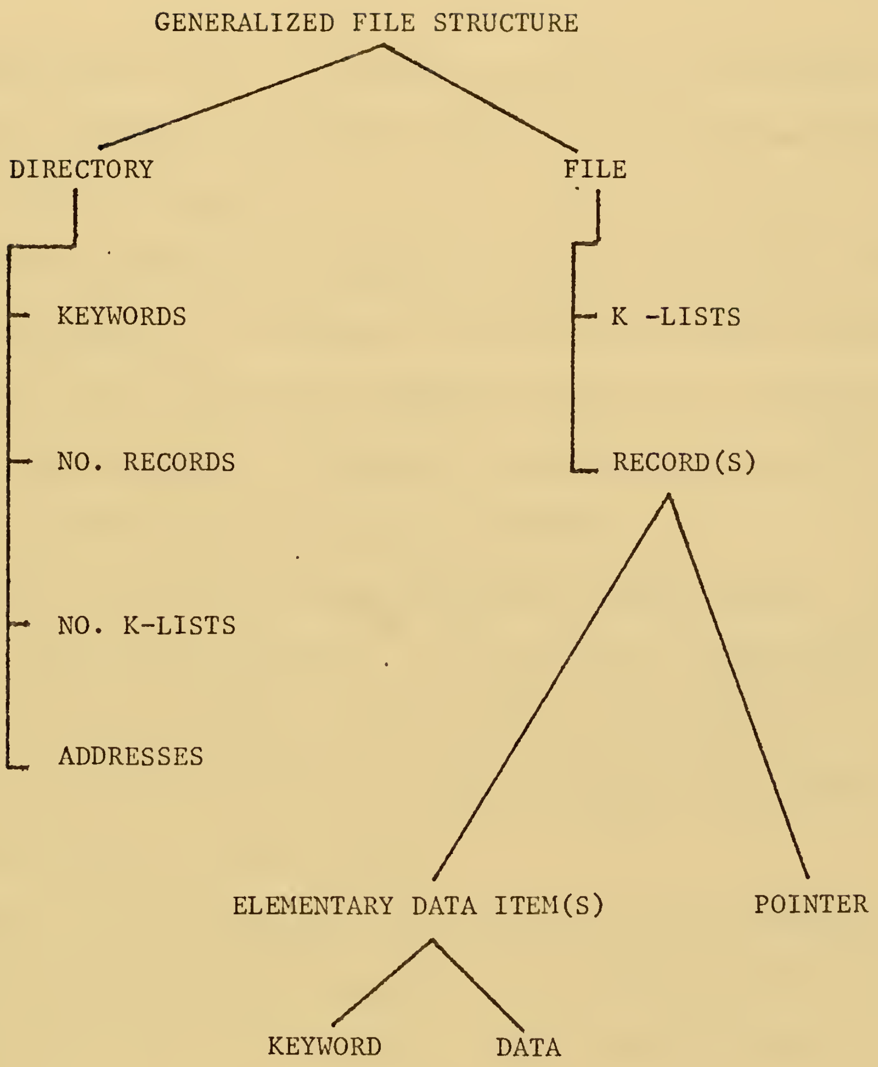


Figure 3.
Hierarchy of Definitions for a Generalized Structure

The ADDRESS a of a record is represented by a positive integer and indicates the location of the record in some type of storage media. Each record has a unique address. In Figure 2 the unique addresses of the respective records are shown in the file by 002, 007, 011, 121 and 145.

A HASH ADDRESS $f(K_i)$ is an address derived by transforming a keyword K_i by a function f, such that $f(K_i) = a_i$. For further explanation of this process see page 37.

A record may contain an elementary data item, called the K-POINTER of R. The pointer is the address of another record which contains the same keyword. The null pointer indicates the end of a sequence of K-pointer linked records. In Figure 2 the elementary data items 121 and 145 are pointers and 0 is the null pointer.

A K-LIST is a set of records containing a common keyword. The list may contain only one record. Also there may be associated with each keyword several K-lists. In one K-list the K-pointers only point to records within that K-list. As shown in Figure 2, the records at addresses 002 and 121 form a K-list.

A FILE F is a collection of records with the same elementary data items. Every K-list containing one or more of these records must be contained within the file. In Figure 2 each record is made up of the same four elementary data items: last name, rank, MOS and a pointer. These records are linked by means of K-pointers into K-lists. Each K-list within the file represents those records containing a common last name keyword.

A DIRECTORY D for a file is a set of sequences of the form

$$(K_i, h_i, n_i; a_{i1}, a_{i2}, \ldots, a_{in_i}) \text{ for } i = 1,2,\ldots,m.$$

the elementary data items within each sequence represent respectively K_i, the i-th keyword; h_i, the number of records containing keyword K_i; n_i, the number of K-lists for each K_i within the file; and the beginning address a_{ij}, of the j-th K_i -list.(5) For an example see the sequences containing last name, h_i, n_i and the addresses in the rectangle marked directory of Figure 7.

A GENERALIZED FILE STRUCTURE consists of two items, a file F with its directory D. Figure 2 is an example of a generalized file structure, as are those files studied in this paper.

B. FILE STRUCTURES

1. Sequential Organization

In a sequential file structure, for every keyword K_i, $h_i = n_i = 1$ and $a_i < a_2 < \ldots < a_m$.(5) For example, if the last name is chosen as the keyword then records would be stored contiguously in alphabetical order according to last name. The records in the file are indicated in the form of a 1-1 correspondence between them and the directory sequences, as there is one keyword K for each record. See Figure 4.

2. Multilist Organization

In a multilist file structure there exists one K-list per keyword, that is every $n_i = 1$.(5) In this file a record R is a member of a K_i-list whenever R contains the keyword K_i. The directory sequences of the multilist form a 1-1 correspondence with the K_i-lists. Only the beginning address of the K_i-list a_{i1}, occurs in the directory. Successive records within the K_i-list are obtained by means of the K-pointer of R, with the null pointer terminating the sequence. See Figure 5. Referring now to Figure 2, Jones would be the keyword by

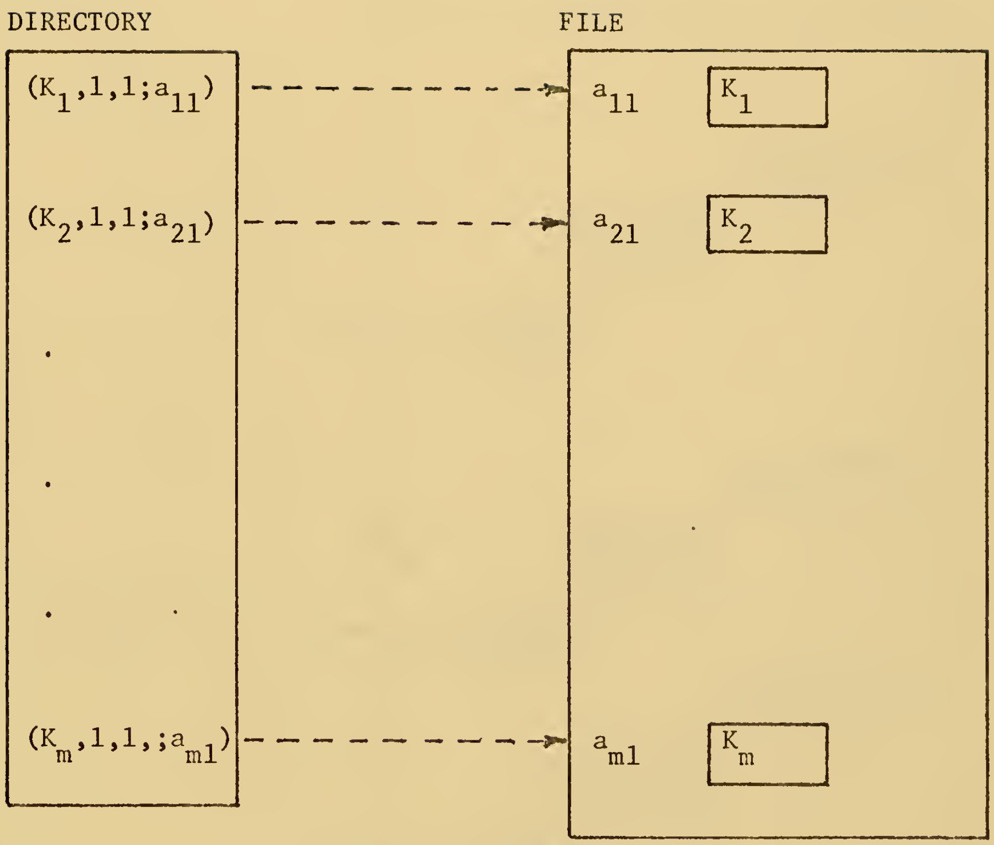


Figure 4.
Sequential Organization

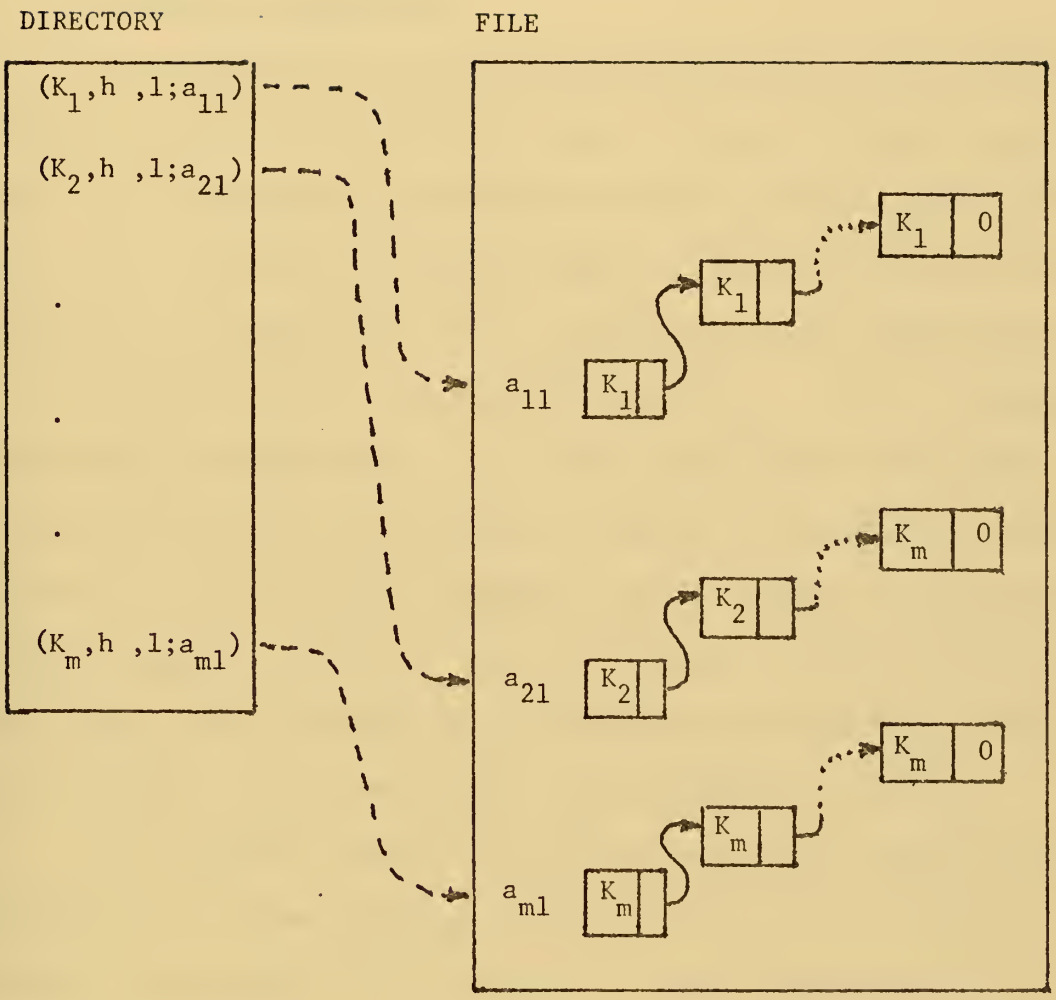


Figure 5.
Multilist Organization

which all records containing the last name Jones are referenced. The directory would contain only one occurrence of the keyword Jones and a single address of a record with this keyword. Subsequent records in the Jones K-list are linked by means of pointers.

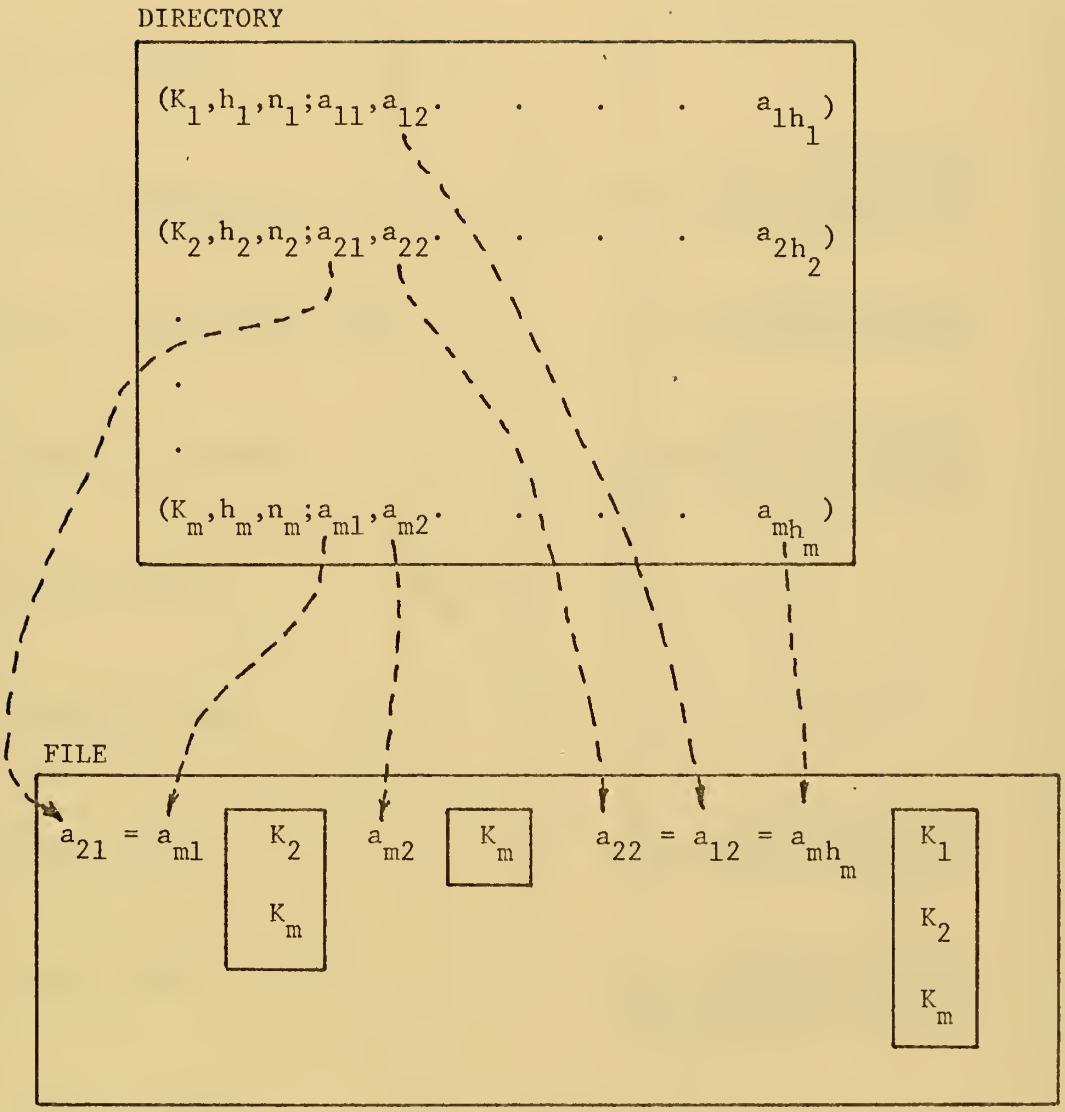
3. Inverted Organization

In an inverted file structure each elementary data item contained within the record R is designated a keyword K within the directory D, such that every K-list contains one and only occurrence of R; that is $h_i = n_i$ for all i.(5) See Figure 6. The directory of an inverted file is usually quite large, because for every keyword there is an associated sequence of record addresses a_{ij}. Thus by assigning each keyword the addresses of all those records which contain the common attribute value, one need only locate the addresses associated with any K_i in the directory to find a set of records containing the common reference. Because of this association, a record address may appear many times throughout the directory in the many K_i-a_{ij} associations. Figure 7 shows how Figure 2 would appear if inverted.

In a partially inverted file structure only a subset of the elementary data items contained within the record are selected as keywords. This type of file structure is often substituted for the "fully" inverted file when known access or retrieval requirements are based solely upon selected attributes.

4. Random Organization

A random file structure is a variation of a generalized file structure in that a directory of keywords does not exist. Instead the keywords are transformed into addresses, these in turn form a listing which can be thought of as a directory. In this file organization a



DIRECTORY
$(K_1, h_1, n_1; a_{11}, a_{12} \cdot \cdot \cdot \cdot a_{1h_1})$
$(K_2, h_2, n_2; a_{21}, a_{22} \cdot \cdot \cdot \cdot a_{2h_2})$
$(K_m, h_m, n_m; a_{m1}, a_{m2} \cdot \cdot \cdot \cdot a_{mh_m})$
FILE
$a_{21} = a_{m1}$
K_2
K_m
a_{m2}
K_m
$a_{22} = a_{12} = a_{mh_m}$
K_1
K_2
K_m

Figure 6.
Inverted Organization

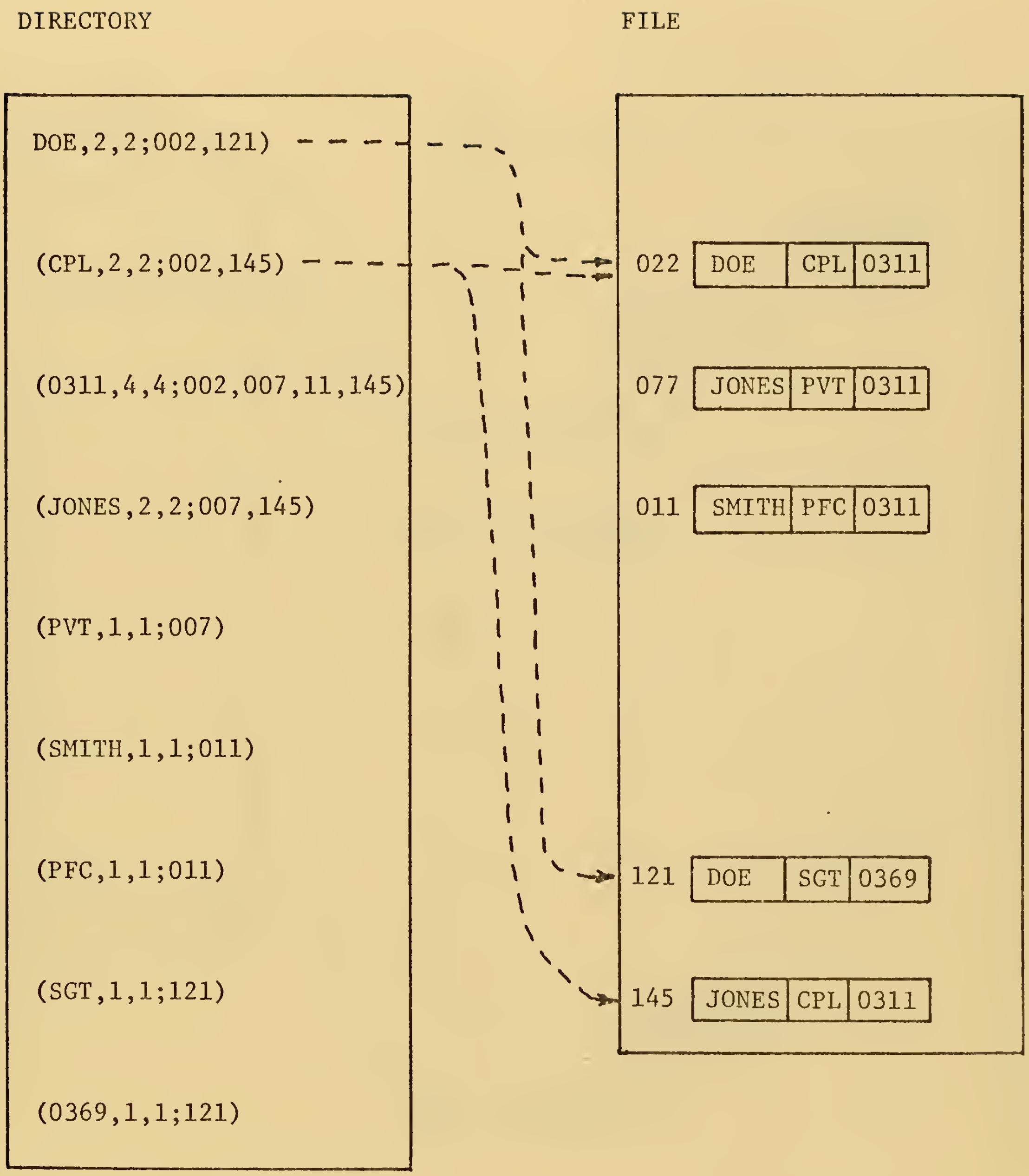


Figure 7.
Figure 2 as an Inverted File Organization

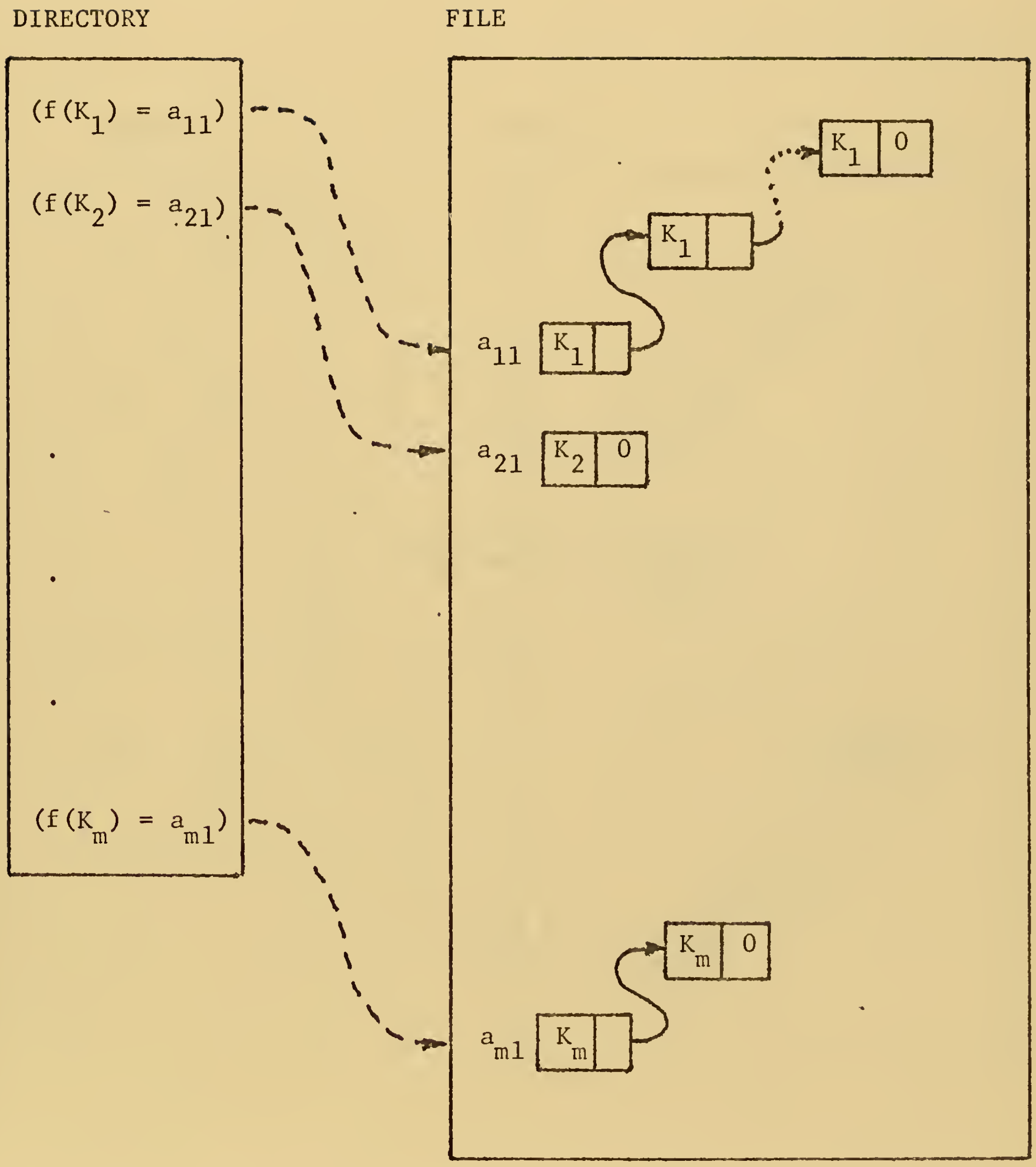


Figure 8.
Random (Calculation) Organization

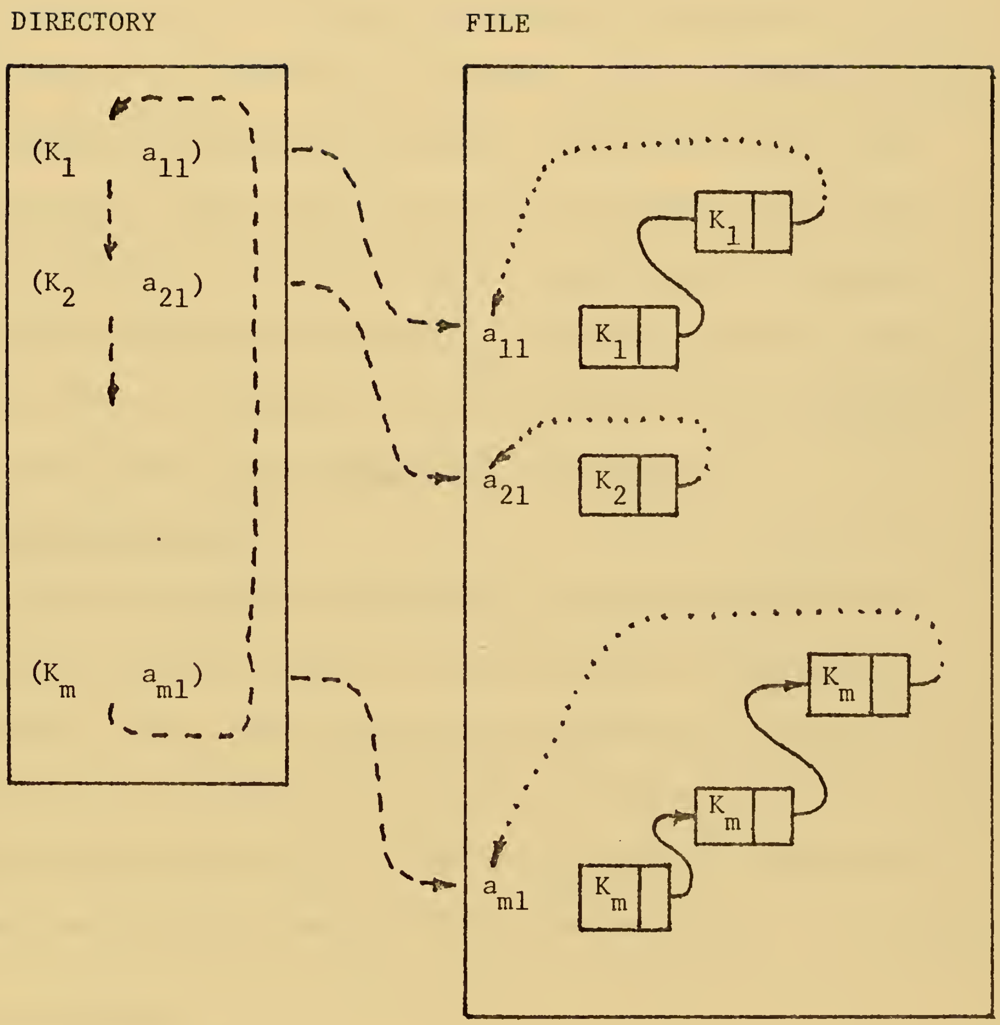


Figure 9.
Ring Organization

record R is stored and retrieved on the basis of a predictable relationship between exactly one keyword K of a record and the address a of the record. This relationship consists of a process of transforming or "hashing" a keyword K of a record R into a numeric address. The calculation process used in determining the record keyword address relationship transforms the keyword into a numeric value by an algorithm chosen for its effective strategy in optimizing storage space. The numeric result is then divided by a divisor, which is predicated on the number of available directory addresses. The remainder after the division becomes a direct entry point to the directory. Since this method is non-perfect, a "collision" will sometimes occur, whereby different keywords map into the same K_i directory location. When such a collision occurs, a K-pointer of R is established from a designated record in the K_i-list to the new record. See Figure 8.

5. Ring Organization

A ring file structure consists of a multilist organization with one major difference, there is no null pointer terminating the K-list sequence. What would normally be considered as the null pointer with respect to K, instead is designated a K-pointer of R to the beginning address of the K_i-list, a_{i1}. Thus the K_i-list of a ring may be continuously and totally traversed from any record within. See Figure 9.

C. SEARCH TECHNIQUES

1. General

Any directory sequence of a file may be defined

$$\alpha_1 = (K_i, h_i, n_i; a_{i1}, a_{i2}, \ldots, a_{in_i}).$$

There is a function f which specifies how the beginning addresses of

K_i-lists in α_i may be traversed.(5) The function's domain is the keyword K_i together with a variable address x in α_i. The range of the function is a single address y in α_i. The null address of both x and y may be specified by 0. Thus

$$y = f(K_i, x)$$

where

$$y = \begin{cases} \min_j(a_{ij}), & x = 0; \\ 0. & x = \max_j(a_{ij}); \\ \min_j(a_{ij} : a_{ij} > x). & \text{otherwise.} \end{cases}$$

Each K_i in the directory of a generalized file structure is associated with n_i distinct beginning addresses. The process of generating these beginning addresses is called decoding the keyword K_i. In order that a keyword K_i be decoded, the function f must be applied first to the initial value of the variable address x so as to produce the beginning address of the first K-list. Then successive applications of the function to the address most recently determined produce each subsequent, higher K-list address. Finally, when the function produces a null address, the so called decoding process for the keyword K_i ceases, as all K_i-lists have been determined. Thus by beginning with x = 0 and applying the function to successive values of y until the null address is reached, the decoded values are

$$f(K_i, 0) = a_{i1}$$

$$f(K_i, a_{i2}) = a_{i2}$$

$$\vdots$$

$$f(K_i, a_{i,n-1})\ a_{in}$$

$$f(K_i, a_{in}) = 0.$$

referring to Figure 7, the initial application of the directory function

$$f(DOE,0) = 002$$

produces the address of the first record in the initial Doe K-list, this being located at address 002. The next application of the function

$$f(Doe,002) = 121$$

produces the first address of the record beginning the second Doe K-list. With one more application of the function

$$f(Doe,121) = 0$$

the null address is determined and the traversing process ceases for the keyword Doe as all K-lists with this keyword have been located. It should be noted that in this example each Doe K-list contains only one record.

There is also a function g for any file F which specifies how each element of a K-list may be traversed.(5) The domain of g is the cartesian product (K x a) of the set K of all keywords in F with the set a of all addresses in F. The range of g is in a. Thus

$$y = g(K_i,x)$$

where y is the K_i-pointer of the record whose address is x. In order that a record R be retrieved, the K_i-pointer of R must have been produced by the function g. In other words, only if its address has been used by the function g for the production of a pointer can a record be considered to have been retrieved. Once again using the example illustrated in Figure 2, the first application of the traversing function produces the address of the first record in the Doe K-list, 002. Then by applying g to this address and the same keyword Doe,

$$g(Doe,002) = 121$$

The address of the next record in the K-list is retrieved. Finally,

$$g(\text{Doe},121) = 0$$

indicates the absence of any more records in the Doe K-list.

2. Sequential and Multilist Organizations

The search technique used for the sequential and multilist organizations consists of a keyword by keyword search through the file directories for a unique K_i. The directory of the sequential file contains one keyword per record in the file. The multilist directory contains one unique keyword for each set of records containing that particular keyword. The search actually consists of a logical comparison of each K_i within the directory (for $i = 1...m$) with the search keyword (K') of the record(s) desired to be retrieved. When $K' = K_i$ then the K_i-list of all records containing K as a keyword is traversed and retrieved. These steps are accomplished by application of the traversing function g:

$$g(K_i,a_{ij}) = R.$$

In the sequential file K' must be compared to subsequent keywords until $K' \neq K_i$ (for $i = 1...m$) to en ure all records containing K' are retrieved.

3. Partially Inverted Organization

Two search techniques are employed with the partially inverted organization, an index-sequential technique and a binary search technique. The directory and file contents are the same for each technique. The file is partially inverted on three separate elementary data items. In turn, the directory is indexed according to each of these three items for immediate access should that particular data item be chosen as the search keyword (K').

The index-sequential search for keyword K_i proceeds as described above for the sequential and multilist file organizations with the additional capability of being able to directly access the subset of directory keywords corresponding to that of K'. Likewise, the K_i-lists of all records containing K_i and concomitant records are traversed and retrieved respectively by appropriate application of the traversing functions.

In the binary search for keyword K_i, the traversing function f is not used, but rather, the directory is sampled in the middle for $K' = K_i$. If $K' > K_i$ then the first half of the keywords in the directory are eliminated from further comparision; if $K' < K_i$ then the latter half of the directory is eliminated. The remaining half of the directory is then sampled again and the process repeated until $K' = K_i$. K_i-lists are traversed and records retrieved as described for the index-sequential technique.

4. Random Organization

The search techinque used for the random organization consists of a two phase process. The first phase consists of transforming the search keyword K' into a hashing address, f(K'). The second phase of a series of logical comparisons of the ordered elements in the K-list associated with f(K') until the appropriate record(s) are retrieved. See page 37 for further amplification of this process.

5. Ring Organization

The search technique used for the ring organization is the same as that used for the multilist organization discussed on page 26.

III. APPLICATION SUBROUTINES

A. GENERAL

A tactical command and control system requires not only the ability to gather data and retrieve it selectively, but also to make this data available to system application programs, i.e. Fire Mission Analysis program. Such a system cannot be solely for the storage and retrieval of rigidly formatted data, but rather it must be capable of answering information needs by supplying facts which may depend upon complex interrelationships within the data. The system normally provides a rationale for structuring data and a means for managing and querying the data base. For purposes of this paper the action of querying the data base is the search and retrieval application, collectively referred to as application subroutines.

User programs and data remain as independent resources to be combined as the need arises. The system maintains information about the location of the data in the file directories. It also maintains information about the input and output requirements of the user's program and has the ability to transform the existing data to meet the requirements of the user's program.

Any tactical data handling system must be capable of working in response to user commands. The user treats his program requirements as a set of operators. The data base is treated as a set of operands to be bound to the operators by means of various application programs, which may be either lengthy processes that consist of many tasks to be executed over large files of data or simple functions that consist of a single operation on a small unit of data.

Typical and ubiquitous to all system applications are the requirements for the storage and retrieval of data. Because of their vital necessity to all data manipulation processes and their commonality to all application programs, the five application subroutines listed below were selected for use in the file structure analysis for this paper.

B. HIGH ACCESS

A high access application subroutine is defined as a single application in which 60 percent or more of the records contained within a file are accessed for the purpose of executing some type of operation. For high access applications both the directory keywords and search keywords are ordered alphabetically or numerically as the situation warrants. Access to a particular record within the file is accomplished by means of the search techniques discussed in the previous section. Elementary data item acquisition and user operations are performed in compliance with specific user program requirements. Other file operations, such as additions and deletions of records, are not evaluated. The number or type of operations to be performed on each record are not considered, as the analysis is concerned specifically with those basic machine operations necessary only to locate a particular record.

C. MEDIUM ACCESS

A medium access application subroutine is defined as a single application in which less than 60 percent and more than 30 percent of the records contained within a file are accessed for the purpose of executing some type of operation. The foregoing considerations for high access are also included.

D. LOW ACCESS

A low access application subroutine is defined as a single application in which less than 30 percent of the records contained within a file are accessed for the purpose of executing some type of operation. The foregoing considerations for high access are also included.

E. SINGLE KEY ACCESS

All records contained within a file having a common keyword are accessed for the purpose of performing some additional application. There may be only one such record or many.

F. MULTIPLE KEY ACCESS

A multiple key access is defined as access for all records within a file which have two or more keywords in common.

IV. EXPERIMENTAL PROCEDURE

A. GENERAL

1. Data Base

The data base used for this paper consisted of 1657 personnel type records. Included in each record were the individual's full name and four interest codes, which were represented by a three digit number. The size of the data base used in this study can be compared to any one of the many groupings of data contained in the MTACCS data base, such as the Decision Logic Table. Therefore, the statistics gathered are representative of data that might be taken from an actual command and control system.

2. Keywords

For purposes of the single attribute file directories, the last name elementary data item in each record was chosen as the keyword. For purposes of the inverted type file directories, three elementary data items in each record were chosen as keywords. They were the individual's last name and two interest codes, interest-1 and interest-4. The selection of these particular keywords was made on the basis of uniqueness and variability. Last names were the most unique with 957 different ones, while interest-4 only had 28 different values. Interest-1 had 153.

3. Building Data Structures

The data Structures were defined in the higher level computer programming language, ALGOL. ALGOL was chosen because of its facility with list processing, which is used extensively with generalized file structures.

Upon completion of construction of the five file structures defined on page 15 (sequential, multilist, partially inverted, random and ring), a series of application subroutines were executed on each in order to compare the file's responsiveness to the different applications.

4. <u>Data Sets</u>

Six data sets of search keywords were organized. One set was arranged randomly. The other five were subsets of the available file and the last name keywords were ordered alphabetically and interest codes numerically.

5. <u>Comparing Structures</u>

In order to compare the responsiveness of the different file structure organizations it was necessary to quantify the different data file basic machine operations, such as logical compare, add, multiply, and divide. No attempt was made to determine the assembly language instructions, which would actually be used in the operation of the IBM 360/67 for the execution of a particular application. It was not considered necessary to perform this analysis, since ALGOL was only used as a representative data base language. Also, since every machine and every language will execute these basic machine operations in a slightly different manner it was decided to keep the analysis at a level that would be common regardless of the machine or language used.

Once the records were retrieved during the application subroutine runs, no additional operations were performed such as adding, deleting or updating the records. The purpose of this paper was served by simply determining the number of various basic operations required to retrieve a data record.

6. Quantifying Procedure

The basic machine operation counting procedure, also known as quantifying procedure, consisted of a two step process, the directory search count and the record retrieval count. The directory search count for all search techniques, except the binary and random, consisted of two logical comparison basic machine operator counts each time a search keyword K' was compared to a keyword K_i. The first logical comparison count was required to check for the end of file, the last entry in the directory. The second logical comparison count was for each search keyword comparison ($K' = K_i$) with the keyword in the directory. The binary directory search count consisted of one add, divide and logical comparison count each and either a subtract or another add count for each occurrence of sampling and halving the directory entries. The random search count consisted of a series of logical comparison, multiply, add and divide counts for each of the necessary steps required by the transformation function (hashing algorithm) used for each search keyword. See page 37 for an explanation of this procedure.

Once the proper keyword was located by the respective directory search technique the record or records associated with that keyword had to be retrieved. The record retrieval count process for all file organizations except the sequential consisted of a single logical comparison count for each record retrieved. This count was required to check for the end of file, this being the null pointer or K-pointer to the beginning record of the K_i-list in the case of the ring organization. The sequential organization required no additional basic operator count once the desired keyword was located, as each keyword formed a 1-1 correspondence with each record in the file.

7. Normalizing Procedure

Since the comparison dealt with different types of basic machine operators it was decided to normalize these operations and base them all on the logical compare value. For purposes of normalization the following values were used:

Basic Machine Operator	Value
Logical Compare	1
Add	1
Multiply	7
Divide	10

The above values were considered to be representative of the relative differences in time of execution for most all machines and languages.

B. FILE STRUCTURING PROCEDURES

1. Sequential

The sequential file structure program shown on page 50 consisted of a directory called K which was an array containing pointers to alphabetically ordered keywords, composed of record last name elementary data items, and a file of logically ordered records. The physical contiguous aspects of sequential files were thereby simulated, that is logical a_1 < logical a_2 < ...logical a_m.

The directory and file construction process proceeded as follows:

(1) As each separate record was read it was decomposed into its elementary data items. A subroutine called ADD was then envoked to add the record to the sequential file and the keyword to the directory.

(2) Each new keyword was compared to the other directory entries to determine its logical position therein. By use of a linked list structure the directory was ordered.

(3) Once the keyword's appropriate logical directory position was established, the record was constructed and its address placed in the directory.

(4) After all records were read and the file and directory established, the last name keywords in the directory were alphabetically ordered and their addresses were then assigned to the array K in this sequence. This process simulated the physical ordering of the file.

2. Multilist

The multilist file structure program shown on page 54 employed a linked list directory called KEY and a file of records organized into K-lists. The directory consisted of alphabetically ordered keywords composed of a single occurrence of all record last name elementary data items, the initial record address of the K_i-list, and a pointer to the next keyword in the directory, K_{i+1}.

The directory and file construction process proceeded as follows:

(1) As each separate record was read it was decomposed into its elementary data items. A subroutine called ADD was then envoked to add the record to the file and the keyword to the directory.

(2) Each new keyword was entered into the directory in alphabetical order. If the keyword was already a member of the directory then no insertion was required and the process branched to step (3).

(3) The record was constructed with all elementary data items with the exception of the last name. The record address was then linked to the K_i-list for that keyword.

3. Partially Inverted

The partially inverted file structure program shown on page 58 consisted of a three section, nine array directory and a file of records.

The first section of the directory consisted of keywords, respectively ordered alphabetically or numerically according to last name or interest code. They were stored in the keyword arrays KN, K1 and K4, which provided the means for indexing the directory according to last name and interest codes. The second section consisted of three address arrays AN, A1 and A4, which contained the record addresses to the single record K_i-lists associated with each keyword. The third section also consisted of three arrays HN, H1 and H4 and contained h_i, the number of record addresses associated with each keyword.

The directory and file construction process proceeded as follows:

(1) As each separate record was read it was decomposed into its elementary data items and the record constructed therefrom. A subroutine called ADD was then envoked to add the keywords and their concomitant K_i-list address and h_i to the directory. This process involved three iterations of the steps below, one iteration for each of three keywords on which the file was inverted.

(2) Each new keyword was categorized according to its attributes. It was then compared to the directory entries within the indexed portion of the directory corresponding to its attribute category. This was done to determine the keyword's position in the directory. Once determined, following keywords were each relocated one array position higher to make the necessary room for the new keyword being inserted. If the keyword was already a member of the directory then no insertion was required and the process branched to step (3).

(3) The address of the record was then linked to the K_i-list of addresses maintained by the address arrays and h_i was incremented by one to reflect this latest record address addition.

4. Random

The random file structure program shown on page 68 employed the calculation technique to locate the keyword. A separate data area was used for the purpose of storing collision overflow records. See page 37. A "directory" consisting of an array of pointers was established for each of the three keywords selected (last name, interest-1, and interest-4); each "directory" was named respectively HASH1, HASH2, and HASH3. No particular hashing strategy was utilized to optimize the storage area. Instead a very straight forward technique was employed.

The transformation function (hashing algorithm) used to determine the keyword address in the array was as follows:

(1) Each alphanumeric symbol in the keyword was matched against a string named ALPH containing all possible symbols.

(2) Upon determining a match, the position of the symbol in the string ALPH was multiplied by the position of the symbol in the keyword.

(3) The product of step (2) was successively summed with the preceding values derived from the same keyword. These values were placed in the variable named TOTAL. For example, the name Jones would result in the following operations:

LTR	STRING POSITION	WORD POSIT	RESULT	TOTAL
J	10	1	10	10
O	15	2	30	40
N	14	3	42	82
E	5	4	20	102
S	19	5	95	197

(4) Upon the completion of summing for all symbols in the keyword the final value was divided by a positive number. This number was named HASHD1, HASHD2, and HASHD3 for the keywords name, interest-1 and interest-4, respectively. The divisor was selected on the basis of the number of storage locations allocated to contain the addresses for each directory. In the example above the divisor was 1657, therefore, the remainder after the division, the hash address, becomes 197.

(5) If two or more keywords should hash to the same location in the array (a collision), a separate data overflow area was established by the method of chaining.

In building the file structure, each new record was read and the hash address calculated for all three keywords. The necessary directory entries were made for each of the three keywords and their associated addresses, or in the case of a collision, the next record address in the chain was established before the next record was read. Therefore, no redundancy of data existed.

5. Ring

The ring file structure program shown on page 73 employed three circular rings which were referred to in the program as NAMERING, I1RING, and I4RING. These rings served as directories. The keywords in each of the three rings were ordered either alphabetically or numerically and were formed in a circularly linked list. Thus each keyword was linked to the next with the last keyword being linked to the starting keyword. Additionally, the associated K-list for each keyword was linked either alphabetically or numerically in circular form.

C. APPLICATION SUBROUTINE PROCESSING PROCEDURES

1. Volume Access Application

The volume access application subroutine procedures in which a relatively large number of records were retrieved during a single application run consisted of the high, medium, and low access application subroutines. The data sets of search keywords used for these application runs were sorted alphabetically and numerically prior to processing. This step was taken because it depicted more realistically the manner in which a tactical command and control subsystem would actually accomplish such a task; that is by means of batch mode operations.

When processing the volume application subroutines it was not necessary to begin each keyword search of the directory with the first keyword of the directory K_1. Instead, as each keyword was located in the directory its position was noted, i.e. K_i. The search for the next keyword began at position K_{i+1}, thereby eliminating any requirement to search again previously searched keywords. This sequential search technique was made possible by the preordering of the data sets of search keywords.

Six separate application subroutine runs were executed each with the five differently ordered data sets. These provided a sample size upon which statistical conclusions are drawn. The sequential and multilist organizations were constructed so that a directory search could only be made using the last name keyword. Because of this any search for an elementary data item other than last name would require two logical comparisons of each of the 1657 records in the file, resulting in an exorbitant total of 3314 comparisons. For this reason no count was made of the basic machine operations necessary to search for any keywords other than the last name.

a. High Access

Run-1 was performed with 575 alphabetically ordered last name search keywords. Run-2 used these same last names plus 92 interest-1 and 17 interest-4 numerically ordered interest code search keywords. Basic machine operation counting and normalizing were performed in accordance with the procedures defined on page 33.

b. Medium Access

Run-3 used the 288 last names, run-4 included 46 interest-1 and 9 interest-4 codes. The alphanumeric ordering, the counting and the normalizing procedures were identical to those utilized in the processing of the high access category.

c. Low Access

Run-5 used the 96 last names, run-6 included 16 interest-1 and 3 interest-4 codes. The alphanumeric ordering, the counting and the normalizing procedures were identical to those utilized in the processing of the high access category.

2. <u>Single Purpose Access Application Subroutines</u>

Single purpose access application subroutines were designed to operate in a non-batch mode as a means of selectively accessing certain particular items for the accomplishment of one primary objective. It should be noted that a limit of three search keywords was imposed in this paper. Subroutines were subdivided into two categories: single key access and multiple key access. An example of a single key access might be a situation where it was desired to access all records containing a particular interest code. A multiple key access might be occasioned by a requirement to access all records which contain two or more search keywords.

The single application subroutines were chosen because they best simulated that type of operation which would occur most frequently during the execution of a tactical command and control application subroutine. An example of this would be the situation were for given attributes of the target, the data base is searched for the possible weapon choices available for effectively attacking the target.

Four separate application subroutine runs were executed. These runs provided a sample upon which satistical conclusions are drawn. Since the sequential and multilist organizations were limited to directory searches involving only the last name, two runs were executed with these organizations.

a. Single Key Access

The single key access category of single purpose access application subroutines used only one search keyword. Run-7 used as the search keyword, the last name. Run-8 used as search keywords a mixture of last name and interest-1 and interest-4 codes. In both runs a sample size of 800 was used. The quantifying and normalizing procedures discussed on page 33 were used and the results are presented in Figure 10.

b. Multiple Key Access

The multiple key access category of single purpose access application subroutines utilized two keywords with which to search. Run-9 searched for all records containing a certain last name and interest code. The run included 100 samples of this operation. Each record contained all elementary data items, so as each record was located, based on the first search keyword, it was then checked for the presence of the second keyword. Thus the search was identical to the single

key access, except for an additional logical comparison which was necessary to check for the presence of the second keyword.

Run-10 searched for all records containing a certain last name or interest code. In this run it was necessary to fully search out both search keywords. One hundred samples of this operation were included in the run.

Both runs used the quantifying and normalizing procedure discussed on page 33. The results of these runs are presented in Figure 10.

V. PRESENTATION OF DATA

With each combination of run and file type in the volume access application subroutines five data sets were executed, thus 150 programs of this type were run gathering data. From each of these groups of five a mean was computed and then divided by the number of keywords searched for in the directory. In the single purpose access application subroutines 20 programs were run gathering data and the results were divided by the number of keywords that were searched for in the directory.

The results of these computations yielded the average number of basic machine operations required per keyword search. These values are presented in Figure 10.

	FILE ORGAN / APPLICATION SUBROUTINE	SEQ	ML	PARTIALLY INVERTED		RANDOM	RING
				SEQ	BIN		
	RUN-1	4	4	10	140	67	7
	HIGH						
V	RUN-2	X	X	14	136	76	9
O	RUN-3	5	7	10	140	67	7
L	MEDIUM						
U	RUN-4	X	X	14	135	74	9
M	RUN-5	6	9	13	140	66	9
E	LOW						
	RUN-6	X	X	17	135	74	11
S	RUN-7	1997	805	1207	139	67	805
I	SINGLE KEY						
N	RUN-8	X	X	1174	129	77	782
G	RUN-9	1596	958	1192	136	72	958
L	MULTIPLE KEY						
E	RUN-10	X	X	1692	252	170	1116

Figure 10.
Number of Basic Machine Operations Required Per Keyword Search

VI. USE OF RESULTS IN DATA STRUCTURE DESIGN

The primary purpose of the Marine Corps MTACCS test bed located at a Camp Pendleton is to develop the specific operational requirements for the MTACCS subsystems. The test bed system hardware and software consist of off-the-shelf commericially available wares. Consequently the result is a rather slow and inefficient system. The Marine Corps is already considering the purchase of a faster computer for the test bed. This is due to the fact that response times for application programs have been poor. In addition to operating system perculiarities, file organization is a major cause of this system's slowness. However, as yet no major effort has been made by test bed personnel to optimize software aspects of the future MTACCS system, such as file organization.

In the process of analyzing the difference between the file organizations studied in this paper it became obvious that while file organization can make major differences in searching efficiency, the organization of the individual record elementary data items can be even more fundamental to overall system searching efficiency.

It is blatantly obvious that minimizing the number of fields that must be searched as keywords improves the effectiveness to search a file. An example of the manner in which an inefficiency in the designing of records impedes file searching is the Decision Logic Table incorporated within the MIFASS data base. This particular data structure is searched by accomplishing an index-sequential search on one keyword. When all associated records have been extracted, five additional

sequential searches are made in which any record that does not contain the remaining keywords is eliminated. This process requires many logical comparisons and is inherently slow.

Files should be scrutinized for possible ways to optimize their use. For example, analysis of the decision logic table reveals that it can be organized into two separate, but related, portions. They could be referred to as the search portion and the weapons selection portion. The search portion consists of six elementary data items, all of which could be represented by a relatively small number of bits, in that each item has only to represent a few values. The below list presents the requirements of these elementary data items:

ELEMENTARY DATA ITEM	POSSIBLE VALUES	BITS REQUIRED
Target Type	14	4
Target Sub-Type	74	7
Target Degree of Protection	10	4
Proximity of Friendly Troops	2	1
Anti-Air Artillery Protected	2	1
Target Mobility	3	2
		Total 19

Thus we have shown that with the use of only 19 bits (less then one word) the portion of the record requiring search could be encoded. This would allow one logical comparison per record. Instead of six, in order to locate the desired record. Additionally, the records could be placed in a random organization (hash coded) thereby expediting the search to an even greater degree.

The weapons selection portion provides a list of weapons that will be effective against a particular type target. More substantial in length, this section includes such information as the preferred weapon, ordnance, fuse, weapon category, probability of kill, weapon CEP, and weapon radius of effectiveness. It appears that all entries are repeated numerous times throughout the file thus indicating the desireability of eliminating their redundancy and, consequently, reducing the storage required. A list of pointers may be used to link the search portion of the record to the weapons selection protion. This would appear to simplify the handling of this cumbersome portion of the file.

This is presented as an example of the type analysis that might be performed on files towards the goal of optimizing these structures for the application subroutines to be used. Future thesis work could possibly be accomplished in the following areas:

(1) Analyzing specific MTACCS files

(2) Determining feasibility of file reorganization

(3) Developing new techniques for compacting files

(4) Developing necessary macro instructions for bit string processing.

It should be emphasized that prior to proceeding with research in this area the student should spend several days at the Camp Pendleton test bed to familiarize himself with the MTACCS system in use.

VII. CONCLUSIONS

Before the file organization is determined the organization and structure of the elementary data items in the records of the file must be analyzed in order to optimize the searching efficiency of the file.

The search technique of the sequential file organization was found to be superior per record relationship to the other file organizations when the three volume access application subroutines were applied to the data base. The multilist, partially inverted and ring organizations, all of which lend themselves to sequential searching, functioned in a semi-efficient manner because it was possible to search for all search keywords with only one search pass through the directory. However, the partially inverted file with binary search technique as well as the random file organization responded with an unsatisfactory performance when subjected to the volume access application subroutines. This was due to the fact that the directory search techniques employed could not take advantage of the ordered search keywords and proceed sequentially through the file, but rather a complete application of the search technique cycle was required for each search keyword lookup.

When the search techniques using single purpose access application subroutines were applied to the data base the random organization was found to be superior to the other file organizations. This organization was an order of magnitude better than any of the others, except for the partially inverted file with the binary search technique. This is the organization that would appear most suitable among the ones considered

for a file like the Decision Logic Table where very few table updates are required and the primary processing will be queries. Interestingly, the sequential file organization, while the best for volume accesses was the least effective for single purpose accesses.

SEQUENTIAL ORGANIZATION MAIN PROGRAM

```
RECORD DATA(STRING(13)LNAME;STRING(12)FNAME;STRING(20)STADD;
            STRING(5)ZIP;STRING(7)PHNUM;STRING(2)AGE;STRING(1)SEX;
            STRING(2)GRADE,SCHOOL;STRING(3)INT1,INT2,INT3,INT4;
            REFERENCE(DATA)NEXT);

STRING(25)NAM; STRING(80)CARD; STRING(20)ST; STRING(5)Z; STRING(7)PNUM;
STRING(2)YRS,G,S; STRING(1)MF,BLNK; STRING(3)I1,I2,I3,I4;
STRING(13)LASTN;  STRING(12)FIRSTN;

REFERENCE(DATA) POINT,PTR;  REFERENCE(DATA) ARRAY K(1::1700);

INTEGER I,J,L,M,N,CNTKEY,CNTEST,NUM,ADDCNT,TOTADD,NORM;

N:=1657;
BLNK:=" ";
POINT:=DATA("999","9","9","9","9","9","9","9","9","9","9","9","9",NULL);
FOR J:=1 UNTIL N DO
  BEGIN
    READCARD(CARD);
    NAM:=CARD(0|25);
    ST:=CARD(25|20);
    Z:=CARD(45|5);
    PNUM:=CARD(50|7);
    YRS:=CARD(57|2);
    MF:=CARD(59|1);
    G:=CARD(60|2);
    S:=CARD(62|2);
    I1:=CARD(64|3); I2:=CARD(67|3); I3:=CARD(70|3); I4:=CARD(73|3);
    I:=M:=L:=0;
    WHILE NAM(M|1) ¬= BLNK DO
      BEGIN
        LASTN(M|1):=NAM(M|1);
        M:=I:=M+1;
        IF M = 13 THEN GOTO IT;
      END;
    WHILE M < 13 DO
      BEGIN
        LASTN(M|1):=BLNK;
        M:=M+1;
      END;
    I:=I+1;
    IT:  WHILE L < 12 DO
      BEGIN
        FIRSTN(L|1):=NAM(I|1);
```

```
          I:=I+1;  L:=L+1;
        END;
      ADD(POINT);
    END;

 COMMENT - ESTABLISH INDEX;
PTR:=POINT;
FOR I:=1 UNTIL N DO
  BEGIN
    K(I):=PTR;
    PTR:=NEXT(PTR);
  END;
```

SEQUENTIAL ORGANIZATION ADD SUBROUTINE

```
PROCEDURE ADD(REFERENCE(DATA) VALUE LINK);
  BEGIN
    IF LASTN < LNAME(LINK) THEN
      BEGIN
        POINT:=DATA(LASTN,FIRSTN,ST,Z,PNUM,YRS,MF,G,S,I1,I2,I3,I4,POINT)
        ;  GOTO OUT;
      END;
    WHILE LNAME(NEXT(LINK)) < LASTN DO LINK:=NEXT(LINK);
  TAB1:  IF LNAME(NEXT(LINK)) ¬= LASTN THEN NEXT(LINK):=
         DATA(LASTN,FIRSTN,ST,Z,PNUM,YRS,MF,G,S,I1,I2,I3,I4,NEXT(LINK))
      ELSE BEGIN
        IF FIRSTN < FNAME(NEXT(LINK)) THEN NEXT(LINK):=
          DATA(LASTN,FIRSTN,ST,Z,PNUM,YRS,MF,G,S,I1,I2,I3,I4,NEXT(LINK))
          ELSE BEGIN
            LINK:=NEXT(LINK);
            GOTO TAB1;
          END;
      END;
  OUT:  END ADD;
```

SEQUENTIAL ORGANIZATION SEARCH SUBROUTINE

```
PROCEDURE SEQSRCH(STRING(80) CARD);
  BEGIN
    LASTN:=CARD(0|13);
    CNTKEY:=2;
    FOR I:=1 UNTIL N DO
      BEGIN
        L:=I;
        IF LNAME(K(I)) ¬= LASTN THEN CNTKEY:=CNTKEY+2
        ELSE GOTO TAB2;
      END;
    WRITE("* * ERROR '",LASTN,"' IS NOT A VALID KEYWORD");
    GOTO OUT;
  TAB2:  L:=L+1;
         WHILE LASTN = LNAME(K(L)) DO
           BEGIN
             CNTKEY:=CNTKEY+2;
             L:=L+1;
           END;
         CNTEST:=CNTEST+CNTKEY+2;
  OUT:   END SEQSRCH;
```

MULTILIST ORGANIZATION MAIN PROGRAM

```
RECORD DATA(STRING(12)FNAME;STRING(20)STADD;STRING(5)ZIP;STRING(7)PHNUM;
            STRING(2)AGE;STRING(1)SEX;STRING(2)GRADE,SCHOOL;STRING(3)
            INT1,INT2,INT3,INT4;REFERENCE(DATA)NEXT);
RECORD KEY(STRING(13)LNAME;REFERENCE(DATA)DOWN;REFERENCE(KEY)ACROSS);

STRING(25)NAM; STRING(13)LASTN; STRING(20)ST; STRING(5)Z; STRING(7)PNUM;
STRING(2)YRS,G,S; STRING(1)MF,BLNK; STRING(3)I1,I2,I3,I4;
STRING(12)FIRSTN;  STRING(80)CARD;

REFERENCE(KEY) POINT,PTR;
REFERENCE(DATA) TEMP;

INTEGER I,J,L,M,N,CNTKEY,CNTEST,NUM;
INTEGER CNT1,CNT2,CNT3,CNT4,CNT5;

N:=1657;
BLNK:=" ";
CNT1:=CNT2:=CNT3:=CNT4:=CNT5:=0;
POINT:=KEY("99999",NULL,NULL);
FOR J:=1 UNTIL N DO
  BEGIN
    READCARD(CARD);
    NAM:=CARD(0|25);
    ST:=CARD(25|20);
    Z:=CARD(45|5);
    PNUM:=CARD(50|7);
    YRS:=CARD(57|2);
    MF:=CARD(59|1);
    G:=CARD(60|2);
    S:=CARD(62|2);
    I1:=CARD(64|3); I2:=CARD(67|3); I3:=CARD(70|3); I4:=CARD(73|3);
    I:=M:=L:=0;
    WHILE NAM(M|1) ¬= BLNK DO
      BEGIN
        LASTN(M|1):=NAM(M|1);
        M:=I:=M+1;
        IF M = 13 THEN GOTO IT;
      END;
    WHILE M < 13 DO
      BEGIN
        LASTN(M|1):=BLNK;
        M:=M+1;
      END;
    I:=I+1;
```

```
  IT:  WHILE L < 12 DO
    BEGIN
      FIRSTN(L|1):=NAM(I|1);
      I:=I+1;  L:=L+1;
    END;
  ADD(POINT);
END;
```

MULTILIST ORGANIZATION ADD SUBROUTINE

```
PROCEDURE ADD(REFERENCE(KEY) VALUE LINK);
  BEGIN
    IF LASTN < LNAME(LINK) THEN
      BEGIN
        POINT:=KEY(LASTN,NULL,POINT);
        DOWN(POINT):=DATA(FIRSTN,ST,Z,PNUM,YRS,MF,G,S,I1,I2,I3,I4,NULL);
        CNT1:=CNT1+1;
      END ELSE
        BEGIN
          WHILE (LNAME(ACROSS(LINK)) < LASTN) DO
            BEGIN
              CNT2:=CNT2+1;
              LINK:=ACROSS(LINK);
            END;
          IF (LNAME(ACROSS(LINK))) = LASTN THEN
            BEGIN
              DOWN(ACROSS(LINK)):=DATA(FIRSTN,ST,Z,PNUM,YRS,MF,G,S,I1,
                   I2,I3,I4,DOWN(ACROSS(LINK)));
              CNT3:=CNT3+1;
            END
        ELSE
          BEGIN
            ACROSS(LINK):=KEY(LASTN,NULL,ACROSS(LINK));
            DOWN(ACROSS(LINK)):=DATA(FIRSTN,ST,Z,PNUM,YRS,MF,G,S,I1,I2,
                 I3,I4,NULL);
            CNT4:=CNT4+1;
          END;
        END;
  END ADD;
```

MULTILIST ORGANIZATION SEARCH SUBROUTINE

```
PROCEDURE SEQSRCH(STRING(80) CARD);
  BEGIN
    M:=0;
    CNTKEY:=3;
    PTR:=POINT;
    LASTN:=CARD(0|13);
    WHILE LNAME(PTR) ¬= "99999" DO
      BEGIN
        IF LNAME(PTR) ¬= LASTN THEN CNTKEY:=CNTKEY+2
          ELSE GOTO TAB3;
        PTR:=ACROSS(PTR);
      END;
    WRITE("* * ERROR '",LASTN,"' IS NOT A VALID KEYWORD");
    GOTO OUT;
  TAB3:  TEMP:=DOWN(PTR);
         WHILE NEXT(TEMP) ¬= NULL DO
           BEGIN
             CNTKEY:=CNTKEY+2;
             TEMP:=NEXT(TEMP);
           END;
         CNTEST:=CNTEST+CNTKEY+1;
  OUT:  END SEQSRCH;
```

PARTIALLY INVERTED ORGANIZATION MAIN PROGRAM

```
RECORD REC(REFERENCE(DATA)PTR;REFERENCE(REC)NEXT);
RECORD DATA(STRING(13)LNAME;STRING(12)FNAME;STRING(20)STADD;STRING(5)ZIP
            ;STRING(7)PHNUM;STRING(2)AGE;STRING(1)SEX;STRING(2)GRADE,
            SCHOOL;STRING(3)INT1,INT2,INT3,INT4);

STRING(25)NAM; STRING(13)LASTN,EPTY; STRING(20)ST; STRING(5)Z; STRING(7)
PNUM; STRING(2)YRS,G,S; STRING(1)MF,BLNK; STRING(3)I1,I2,I3,I4,EMTY;
STRING(12)FIRSTN;  STRING(80)CARD; STRING(13) ARRAY KN(1::1000);
STRING(3) ARRAY K1(1::200);  STRING(3) ARRAY K4(1::50);

REFERENCE(DATA) POINT; REFERENCE(REC) ARRAY AN(1::1000);
REFERENCE(REC) ARRAY A1(1::200);  REFERENCE(REC) ARRAY A4(1::50);

INTEGER I,J,L,M,N,T,CNTKEY,CNTEST,CNTN,CNT1,CNT4,TPLUS1,NUM;
INTEGER ADDCNT,TOTADD,NORM,DIVCNT,TOTDIV,SUBCNT,TOTSUB;
INTEGER ARRAY HN(1::1000);  INTEGER ARRAY H1(1::200);
INTEGER ARRAY H4(1::50);

N:=1657;
EPTY:="             ";  EMTY:="   ";  BLNK:=" ";
CNTN:=CNT1:=CNT4:=0;
FOR I:=1 UNTIL 1000 DO
  BEGIN
    AN(I):=REC(NULL,NULL);
    HN(I):=0;
    KN(I):=EPTY;
  END;
FOR I:=1 UNTIL 200 DO
  BEGIN
    A1(I):=REC(NULL,NULL);
    H1(I):=0;
    K1(I):=EMTY;
  END;
FOR I:=1 UNTIL 50 DO
  BEGIN
    A4(I):=REC(NULL,NULL);
    H4(I):=0;
    K4(I):=EMTY;
  END;

 COMMENT - COMMENCE INPUT OF DATA AND FILE CONSTRUCTION;
FOR J:=1 UNTIL N DO
  BEGIN
    READCARD(CARD);
```

```
  NAM:=CARD(0|25);
  ST:=CARD(25|20);
  Z:=CARD(45|5);
  PNUM:=CARD(50|7);
  YRS:=CARD(57|2);
  MF:=CARD(59|1);
  G:=CARD(60|2);
  S:=CARD(62|2);
  I1:=CARD(64|3);  I2:=CARD(67|3);
  I3:=CARD(70|3);  I4:=CARD(73|3);
  IF (I1 = EMTY) OR (I1 > "699") THEN I1:="999";
  IF (I4 = EMTY) OR (I4 < "700") THEN I4:="999";
  I:=M:=L:=0;
  WHILE NAM(M|1) ¬= BLNK DO
    BEGIN
      LASTN(M|1):=NAM(M|1);
      M:=I:=M+1;
      IF M = 13 THEN GOTO IT;
    END;
  WHILE M < 13 DO
    BEGIN
      LASTN(M|1):=BLNK;
      M:=M+1;
    END;
  I:=I+1;
  IT:  WHILE L < 12 DO
    BEGIN
      FIRSTN(L|1):=NAM(I|1);
      I:=I+1;  L:=L+1;
    END;
  POINT:=DATA(LASTN,FIRSTN,ST,Z,PNUM,YRS,MF,G,S,I1,I2,I3,I4);
  ADD(POINT);
END;
```

PARTIALLY INVERTED ORGANIZATION ADD SUBROUTINE

```
PROCEDURE ADD(REFERENCE(DATA) VALUE LINK);
  BEGIN
    IF LASTN = EPTY THEN GOTO TAB3;
    FOR I:=1 UNTIL 1000 DO
      BEGIN
        L:=I;
        IF KN(I) = EPTY THEN
          BEGIN
            KN(I):=LASTN;
            GOTO TAB1;
          END;
        IF KN(I) = LASTN THEN
          BEGIN
            IF PTR(AN(I)) = LINK THEN GOTO TAB3 ELSE GOTO TAB2;
          END;
        IF LASTN < KN(I) THEN
          BEGIN
            FOR T:=CNTN STEP -1 UNTIL I DO
              BEGIN
                TPLUS1:=T+1;
                KN(TPLUS1):=KN(T);
                AN(TPLUS1):=AN(T);
                HN(TPLUS1):=HN(T);
              END;
            KN(I):=LASTN;
            HN(I):=0;
            GOTO TAB1;
          END;
      END;
      TAB1:  CNTN:=CNTN+1;
      TAB2:  AN(L):=REC(LINK,AN(L));
             HN(L):=HN(L)+1;
    TAB3:  IF I1 = "999" THEN GOTO TAB6;
    FOR I:=1 UNTIL 200 DO
      BEGIN
        L:=I;
        IF K1(I) = EMTY THEN
          BEGIN
            K1(I):=I1;
            GOTO TAB4;
          END;
        IF K1(I) = I1 THEN
          BEGIN
            IF PTR(A1(I)) = LINK THEN GOTO TAB6 ELSE GOTO TAB5;
```

```
      END;
    IF I1 < K1(I) THEN
      BEGIN
        FOR T:=CNT1 STEP -1 UNTIL I DO
          BEGIN
            TPLUS1:=T+1;
            K1(TPLUS1):=K1(T);
            A1(TPLUS1):=A1(T);
            H1(TPLUS1):=H1(T);
          END;
        K1(I):=I1;
        H1(I):=0;
        GOTO TAB4;
      END;
  END;
  TAB4:  CNT1:=CNT1+1;
  TAB5:  A1(L):=REC(LINK,A1(L));
         H1(L):=H1(L)+1;
TAB6:  IF I4 = "999" THEN GOTO OUT;
FOR I:=1 UNTIL 50 DO
  BEGIN
    L:=I;
    IF K4(I) = EMTY THEN
      BEGIN
        K4(I):=I4;
        GOTO TAB7;
      END;
    IF K4(I) = I4 THEN
      BEGIN
        IF PTR(A4(I)) = LINK THEN GOTO OUT ELSE GOTO TAB8;
      END;
    IF I4 < K4(I) THEN
      BEGIN
        FOR T:=CNT4 STEP -1 UNTIL I DO
          BEGIN
            TPLUS1:=T+1;
            K4(TPLUS1):=K4(T);
            A4(TPLUS1):=A4(T);
            H4(TPLUS1):=H4(T);
          END;
        K4(I):=I4;
        H4(I):=0;
        GOTO TAB7;
      END;
  END;
  TAB7:  CNT4:=CNT4+1;
  TAB8:  A4(L):=REC(LINK,A4(L));
         H4(L):=H4(L)+1;
```

```
OUT:  END ADD;
```

PARTIALLY INVERTED ORGANIZATION SEQUENTIAL SEARCH SUBROUTINE

```
PROCEDURE SEQSRCH(STRING(80) CARD);
  BEGIN
    I:=1;
    I1:=CARD(0|3);
    ADDCNT:=CNTKEY:=0;
    IF I1 < "100" THEN
      BEGIN
        M:=0;
        LASTN:=CARD(0|13);
        WHILE KN(I) ¬= EPTY DO
          BEGIN
            IF KN(I) ¬= LASTN THEN
              BEGIN
                CNTKEY:=CNTKEY+2;
                I:=I+1;
                ADDCNT:=ADDCNT+1;
              END
            ELSE
              BEGIN
                CNTKEY:=CNTKEY+HN(I)+HN(I);
                ADDCNT:=ADDCNT+HN(I);
                CNTN:=CNTN+1;
                GOTO TABB;
              END;
          END;
        WRITE("* * ERROR '",LASTN,"' IS NOT A VALID KEYWORD");
        GOTO OUT;
      END;
    IF I1 < "700" THEN
      BEGIN
        WHILE K1(I) ¬= EMTY DO
          BEGIN
            IF K1(I) ¬= I1 THEN
              BEGIN
                CNTKEY:=CNTKEY+2;
                I:=I+1;
                ADDCNT:=ADDCNT+1;
              END
            ELSE
              BEGIN
                CNTKEY:=CNTKEY+H1(I)+H1(I);
                ADDCNT:=ADDCNT+H1(I);
                CNT1:=CNT1+1;
                GOTO TABB;
```

```
                END;
            END;
         GOTO TABA;
      END;
   WHILE K4(I) ¬= EMTY DO
      BEGIN
         IF K4(I) ¬= I1 THEN
            BEGIN
               CNTKEY:=CNTKEY+2;
               I:=I+1;
               ADDCNT:=ADDCNT+1;
            END
         ELSE
            BEGIN
               CNTKEY:=CNTKEY+H4(I)+H4(I);
               ADDCNT:=ADDCNT+H4(I);
               GOTO TABB;
            END;
      END;
   TABA:    WRITE("* * ERROR '",I1;"' IS NOT A VALID KEYWORD");
            GOTO OUT;
TABB: CNTEST:=CNTEST+CNTKEY+2;
      TOTADD:=TOTADD+ADDCNT;
OUT:  END SEQSRCH;
```

PARTIALLY INVERTED ORGANIZATION BINARY SEARCH SUBROUTINE

```
PROCEDURE BINSRCH(STRING(13)LASTN);
  BEGIN
    N:=957;
    ADDCNT:=SUBCNT:=DIVCNT:=CNTKEY:=L:=0;
  TABC:  I:=(M+N) DIV 2;
    ADDCNT:=ADDCNT+1;
    DIVCNT:=DIVCNT+1;
    CNTKEY:=CNTKEY+3;
    IF (I = L) OR (I = 0) THEN GOTO TABD;
    L:=I;
    IF LASTN < KN(I) THEN
      BEGIN
        N:=I-1;
        SUBCNT:=SUBCNT+1;
        GOTO TABC;
      END;
    CNTKEY:=CNTKEY+1;
    IF LASTN > KN(I) THEN
      BEGIN
        M:=I+1;
        ADDCNT:=ADDCNT+1;
        GOTO TABC;
      END;
    IF LASTN = KN(I) THEN
      BEGIN
    CNTEST:=CNTEST+CNTKEY+1+HN(I);
        TOTADD:=TOTADD+ADDCNT;
        TOTSUB:=TOTSUB+SUBCNT;
        TOTDIV:=TOTDIV+DIVCNT;
        GOTO OUT;
      END;
  TABD:  WRITE("* * ERROR '",LASTN,"' IS NOT A VALID KEYWORD");
  OUT:  CNTN:=CNTN+1;
  END BINSRCH;

PROCEDURE BI1SRCH(STRING(3)I1);
  BEGIN
    N:=153;
    ADDCNT:=SUBCNT:=DIVCNT:=CNTKEY:=L:=0;
  TABE:  I:=(M+N) DIV 2;
    ADDCNT:=ADDCNT+1;
    DIVCNT:=DIVCNT+1;
    CNTKEY:=CNTKEY+3;
    IF (I = L) OR (I = 0) THEN GOTO TABF;
```

```
      L:=I;
      IF I1 < K1(I) THEN
        BEGIN
          N:=I-1;
          SUBCNT:=SUBCNT+1;
          GOTO TABE;
        END;
      CNTKEY:=CNTKEY+1;
      IF I1 > K1(I) THEN
        BEGIN
          M:=I+1;
          ADDCNT:=ADDCNT+1;
          GOTO TABE;
        END;
      IF I1 = K1(I) THEN
        BEGIN
          CNTEST:=CNTEST+CNTKEY+1+H1(I);
          TOTADD:=TOTADD+ADDCNT;
          TOTSUB:=TOTSUB+SUBCNT;
          TOTDIV:=TOTDIV+DIVCNT;
          GOTO OUT;
        END;
    TABF:  WRITE("* * ERROR '",I1,"' IS NOT A VALID KEYWORD");
    OUT:   CNT1:=CNT1+1;
    END BI1SRCH;

  PROCEDURE BI4SRCH(STRING(3)I4);
    BEGIN
      N:=28;
      ADDCNT:=SUBCNT:=DIVCNT:=CNTKEY:=L:=0;
    TABG:  I:=(M+N) DIV 2;
      ADDCNT:=ADDCNT+1;
      DIVCNT:=DIVCNT+1;
      CNTKEY:=CNTKEY+3;
      IF (I = L) OR (I = 0) THEN GOTO TABH;
      L:=I;
      IF I4 < K4(I) THEN
        BEGIN
          N:=I-1;
          SUBCNT:=SUBCNT+1;
          GOTO TABG;
        END;
      CNTKEY:=CNTKEY+1;
      IF I4 > K4(I) THEN
        BEGIN
          M:=I+1;
          ADDCNT:=ADDCNT+1;
          GOTO TABG;
```

```
     END;
   IF I4 = K4(I) THEN
     BEGIN
       CNTEST:=CNTEST+CNTKEY+1+H4(I);
       TOTADD:=TOTADD+ADDCNT;
       TOTSUB:=TOTSUB+SUBCNT;
       TOTDIV:=TOTDIV+DIVCNT;
       GOTO OUT;
     END;
TABH:  WRITE("* * ERROR '",I4,"' IS NOT A VALID KEYWORD");
OUT:  CNT4:=CNT4+1;
END BI4SRCH;
```

RANDOM (CALCULATION) ORGANIZATION MAIN PROGRAM

```
BEGIN
  STRING(25)NAM,HN,LN,RN;STRING(20)ST;STRING(5)Z;STRING(1)MF,BLANK;
  STRING(2)A,G,S;STRING(3)I1,I2,I3,I4,HI,HZ;STRING(80)CARD;

  INTEGER HASHN1,HASHN2,HASHN3,HASHD1,HASHD2,HASHD3,N,N1,N2,N3,
          COLLISIONS,TOTAL,NUM1,NUM2,NUM3,F,L;
  INTEGER COLLISIONN,COLLISION1,COLLISION4,NUMSEARCH;
  RECORD DATA(STRING(25)LNAME,RNAME;STRING(20)STADD;STRING(5)ZIP;
              STRING(2)AGE;STRING(1)SEX;STRING(2)GRA,SCH;STRING(3)INT1,
              INT2,INT3,INT4;REFERENCE(DATA)PT1;REFERENCE(DATA)PT2;
              REFERENCE(DATA)PT3);
  REFERENCE(DATA)TEMP1,TEMP2,TEMP3;
  REFERENCE(DATA) ARRAY HASH1(0::2000);
  REFERENCE(DATA) ARRAY HASH2(0::2000);
  REFERENCE(DATA) ARRAY HASH3(0::2000);

  INTEGER PROCEDURE GETNUMBER(STRING(1) VALUE LTR);
    BEGIN INTEGER L; STRING(39) ALPH;
    L:=0; ALPH:="ABCDEFGHIJKLMNOPQRSTUVWXYZ 1234567890&/";
    WHILE LTR(0|1)¬=ALPH(L|1) DO
      L:=L+1;
    L+1
    END GETNUMBER;


  COLLISIONS:=0; BLANK:=" ";
  COLLISIONN:=COLLISION1:=COLLISION4:=0;
  N:=N1:=N2:=N3:=1657;
  HASHD1:=HASHD2:=HASHD3:=1657;
HASHN2:=3;
HASHN3:=3;
  FOR M:=0 STEP 1 UNTIL N3 DO HASH3(M):=NULL;
  FOR M:=0 STEP 1 UNTIL N2 DO HASH2(M):=NULL;
  FOR M:=0 STEP 1 UNTIL N1 DO HASH1(M):=NULL;
  FOR M:=1 STEP 1 UNTIL N DO
    BEGIN
    READCARD(CARD);
    NAM:=CARD(0|25); ST:=CARD(25|20); Z:=CARD(45|5); A:=CARD(57|2);
    MF:=CARD(59|1); G:=CARD(60|2); S:=CARD(62|2); I1:=CARD(64|3);
    I2:=CARD(67|3); I3:=CARD(70|3); I4:=CARD(73|3);
    IF I1>"699" THEN I1:="999";IF I4<"700" THEN I4:="999";
    L:=0; LN:=RN:=BLANK;
    WHILE NAM(L|1)¬=BLANK DO
      BEGIN LN(L|1):=NAM(L|1);L:=L+1;END;
```

```
FOR K:=L UNTIL 24 DO
  RN(K|1):=NAM(K|1);
  TOTAL:=0; HN:=LN;
  FOR I:=1 STEP 1 UNTIL L DO
    TOTAL:=TOTAL+GETNUMBER(HN(I-1|1))*I;
  NUM1:=TOTAL REM HASHD1;
  TOTAL:=0; HZ:=I1;
  FOR I:=1 STEP 1 UNTIL HASHN2 DO
    TOTAL:=TOTAL+GETNUMBER(HZ(I-1|1))*I;
  NUM2:=TOTAL REM HASHD2;
  TOTAL:=0; HI:=I4;
  FOR I:=1 STEP 1 UNTIL HASHN3 DO
    TOTAL:=TOTAL+GETNUMBER(HI(I-1|1))*I;
  NUM3:=TOTAL REM HASHD3;

      F:=0;

      TEMP1:=HASH1(NUM1);
      IF TEMP1=NULL THEN BEGIN     F:=F+1     END ELSE
        BEGIN COLLISIONN:=COLLISIONN+1;
        IF HN<LNAME(TEMP1) THEN F:=F+2;
        WHILE(PT1(TEMP1)¬=NULL)AND(LNAME(PT1(TEMP1))<=HN) DO
          TEMP1:=PT1(TEMP1);
        END;
      TEMP2:=HASH2(NUM2);
      IF TEMP2=NULL THEN BEGIN     F:=F+4     END ELSE
        BEGIN COLLISION1:=COLLISION1+1;
        IF HZ<INT1(TEMP2) THEN F:=F+8;
        WHILE(PT2(TEMP2)¬=NULL)AND(INT1(PT2(TEMP2))<=HZ) DO
          TEMP2:=PT2(TEMP2);
        END;
      TEMP3:=HASH3(NUM3);
      IF TEMP3=NULL THEN BEGIN     F:=F+16    END ELSE
        BEGIN COLLISION4:=COLLISION4+1;
        IF HI<INT4(TEMP3) THEN F:=F+32;
        WHILE(PT3(TEMP3)¬=NULL)AND(INT4(PT3(TEMP3))<=HI) DO
          TEMP3:=PT3(TEMP3);
        END;
COMMENT F IS A COUNTER THAT HAS BEEN USED TO DETERMINE WHICH
    CONSTRUCTION TO USE IN THE BUILDING THE RECORD IN THE DATA
    STRUCTURE.;
        IF F=0 THEN GO TO J;
        IF((F=1)OR(F=2))THEN GO TO B;
        IF((F= 4)OR(F= 8)) THEN GO TO C;
        IF((F=16)OR(F=32)) THEN GO TO E;
        IF((F= 5)OR(F=6)OR(F=9)OR(F=10)) THEN GO TO D;
        IF((F=17)OR(F=18)OR(F=33)OR(F=34)) THEN CO TO P;
        IF((F=20)OR(F=24)OR(F=36)OR(F=40)) THEN GO TO K;
```

```
                    IF((F=21)OR(F=22)OR(F=25)OR(F=26)OR(F=37)OR(F=38)OR(F=41)OR(
                       F=42)) THEN GO TO H;
J:PT1(TEMP1) :=PT2(TEMP2) :=PT3(TEMP3) :=DATA(LN,RN,ST,Z,A,MF,G,S,I1,I2,
    I3,I4,PT1(TEMP1) ,PT2(TEMP2) ,PT3(TEMP3));
    GO TO JUMP;
B:HASH1(NUM1):=PT2(TEMP2) :=PT3(TEMP3) :=DATA(LN,RN,ST,Z,A,MF,G,S,I1,I2,
    I3,I4,HASH1(NUM1),PT2(TEMP2) ,PT3(TEMP3));
    GO TO JUMP;
C:PT1(TEMP1) :=HASH2(NUM2):=PT3(TEMP3) :=DATA(LN,RN,ST,Z,A,MF,G,S,I1,I2,
    I3,I4,PT1(TEMP1) ,HASH2(NUM2),PT3(TEMP3));
    GO TO JUMP;
D:HASH1(NUM1):=HASH2(NUM2):=PT3(TEMP3) :=DATA(LN,RN,ST,Z,A,MF,G,S,I1,I2,
    I3,I4,HASH1(NUM1),HASH2(NUM2),PT3(TEMP3));
    GO TO JUMP;
E:PT1(TEMP1) :=PT2(TEMP2) :=HASH3(NUM3):=DATA(LN,RN,ST,Z,A,MF,G,S,I1,I2,
    I3,I4,PT1(TEMP1) ,PT2(TEMP2) ,HASH3(NUM3));
    GO TO JUMP;
P:HASH1(NUM1):=PT2(TEMP2) :=HASH3(NUM3):=DATA(LN,RN,ST,Z,A,MF,G,S,I1,I2,
    I3,I4,HASH1(NUM1),PT2(TEMP2) ,HASH3(NUM3));
    GO TO JUMP;
K:PT1(TEMP1) :=HASH2(NUM2):=HASH3(NUM3):=DATA(LN,RN,ST,Z,A,MF,G,S,I1,I2,
    I3,I4,PT1(TEMP1) ,HASH2(NUM2),HASH3(NUM3));
    GO TO JUMP;
H:HASH1(NUM1):=HASH2(NUM2):=HASH3(NUM3):=DATA(LN,RN,ST,Z,A,MF,G,S,I1,I2,
    I3,I4,HASH1(NUM1),HASH2(NUM2),HASH3(NUM3));
JUMP:END;
END.
```

RANDOM (CALCULATION) ORGANIZATION SEARCH PROGRAM

```
BEGIN INTEGER II,IJ,IK,NASEARCH,I1SEARCH,I4SEARCH,COMPARE;
      INTEGER AKNT,MKNT,DKNT,NKNT,NORM;
    NASEARCH:=96;      I1SEARCH:=0;     I4SEARCH:=0;
    II:=IJ:=IK:=COMPARE:=AKNT:=NKNT:=MKNT:=DKNT:=0;
    WHILE(II<NASEARCH)OR(IJ<I1SEARCH)OR(IK<I4SEARCH)DO
      BEGIN STRING(3)CHKSTR;
          READCARD(CARD);
          CHKSTR:=CARD(0|3);
          IF CHKSTR<"100" THEN
            BEGIN
            IF    II<NASEARCH THEN
            BEGIN
            II:=II+1;
              L:=0; LN:=BLANK;
              WHILE CARD(L|1)¬=BLANK DO
               BEGIN LN(L|1):=CARD(L|1);L:=L+1;END;
              HN:=LN; TOTAL:=0;
              FOR I:=1 STEP 1 UNTIL L DO
                TOTAL:=TOTAL+GETNUMBER(HN(I-1|1))*I;
              NUM1:=TOTAL REM HASHD1;
    AKNT:=AKNT+L;
    DKNT:=DKNT+1;
    MKNT:=MKNT+L;
              TEMP1:=HASH1(NUM1);
              IF TEMP1=NULL THEN
             BEGIN WRITE("ERROR ",CARD); GO TO EXIT; END;
              WHILE(PT1(TEMP1)¬=NULL) AND(LNAME(TEMP1)<=HN) DO
       BEGIN TEMP1:=PT1(TEMP1);COMPARE:=COMPARE+2;END;
              IF(PT1(TEMP1) =NULL)AND(LNAME(TEMP1)=HN)THEN
                  COMPARE:=COMPARE+2;
            END;
            END
            ELSE
          BEGIN
          IF CHKSTR<"700" THEN
            BEGIN
            IF    IJ<I1SEARCH THEN
            BEGIN
            IJ:=IJ+1;
              HZ:=CARD(0|3);  TOTAL:=0;
              FOR I:=1 STEP 1 UNTIL HASHN2 DO
                TOTAL:=TOTAL+GETNUMBER(HZ(I-1|1))*I;
              NUM2:=TOTAL REM HASHD2;
    AKNT:=AKNT+L;
```

```
      DKNT:=DKNT+1;
      MKNT:=MKNT+L;
                TEMP2:=HASH2(NUM2);
                IF TEMP2=NULL THEN
               BEGIN WRITE("ERROR ",CARD); GO TO EXIT; END;
                WHILE(PT2(TEMP2)¬=NULL) AND(INT1(TEMP2)<=HZ) DO
         BEGIN TEMP2:=PT2(TEMP2);COMPARE:=COMPARE+2;END;
                IF(PT2(TEMP2) =NULL)AND(INT1(TEMP2)=HZ)THEN
                    COMPARE:=COMPARE+2;
              END;
              END
              ELSE
              BEGIN
              IF     IK<I4SEARCH THEN
              BEGIN
              IK:=IK+1;
                HI:=CARD(0|3);  TOTAL:=0;
                FOR I:=1 STEP 1 UNTIL HASHN3 DO
                  TOTAL:=TOTAL+GETNUMBER(HI(I-1|1))*I;
                NUM3:=TOTAL REM HASHD3;
      AKNT:=AKNT+L;
      DKNT:=DKNT+1;
      MKNT:=MKNT+L;
                TEMP3:=HASH3(NUM3);
                IF TEMP3=NULL THEN
               BEGIN WRITE("ERROR ",CARD); GO TO EXIT; END;
                WHILE(PT3(TEMP3)¬=NULL) AND(INT4(TEMP3)<=HI) DO
         BEGIN TEMP3:=PT3(TEMP3);COMPARE:=COMPARE+2;END;
                IF(PT3(TEMP3) =NULL)AND(INT4(TEMP3)=HI)THEN
                    COMPARE:=COMPARE+2;
              END;
              END;
            END;
  EXIT:   END;
WRITE(" ");
WRITE(" ");
WRITE("ADD=",AKNT,"MULT=",MKNT,"DIV=",DKNT);
WRITE("LOGICAL COMPARISONS=",COMPARE);
NORM:=AKNT+NKNT+COMPARE+7*MKNT+10*DKNT;
WRITE(" ");  WRITE("NORMALIZED UNITS=",NORM);
    END;
```

RING ORGANIZATION MAIN PROGRAM

```
BEGIN
  STRING(25)NAM,HN;STRING(20)ST;STRING(5)Z,HZ;STRING(1)MF;STRING(2)G,S;
  STRING(3)I1,I2,I3,I4,HI; STRING(80)CARD;STRING(2)A;
  STRING(25)LASTNAM,LN;STRING(1)BLANK;
  INTEGER N,I,L,NUMSEARCH;

  RECORD DATA(STRING(25)NAME;STRING(20)STADD;STRING(5)ZIP;STRING(2)AGE;
              STRING(1)SEX;STRING(2)GRA,SCH;STRING(3)INT1,INT2,INT3,
              INT4;REFERENCE(DATA)PN;REFERENCE(DATA)PT1;
              REFERENCE(DATA)PT4);
  RECORD NAMERING(STRING(25)LASTNAME;REFERENCE(DATA)PTD;REFERENCE
               (NAMERING)PTN);
  RECORD I1RING(STRING(3)IT1;REFERENCE(DATA)PTA;REFERENCE(I1RING)PTB);
  RECORD I4RING(STRING(3)IT4;REFERENCE(DATA)PTC;REFERENCE(I4RING)PTE);
  REFERENCE(NAMERING)NAMPTR,TEMPN;
  REFERENCE(I1RING)I1PTR,TEMP1;
  REFERENCE(I4RING)I4PTR,TEMP4;
  REFERENCE(DATA)TDATA,TEMP;

  N:=1657;
  BLANK:=" "; NAMPTR:=NULL; I1PTR:=NULL; I4PTR:=NULL;
  FOR M:=1 STEP 1 UNTIL N DO
    BEGIN
    READCARD(CARD);
    NAM:=CARD(0|25); ST:=CARD(25|20); Z:=CARD(45|5); A:=CARD(57|2);
    MF:=CARD(59|1); G:=CARD(60|2); S:=CARD(62|2); I1:=CARD(64|3);
    I2:=CARD(67|3); I3:=CARD(70|3); I4:=CARD(73|3);
    IF I1>"699" THEN I1:="999";IF I4<"700" THEN I4:="999";
    TDATA:=DATA(NAM,ST,Z,A,MF,G,S,I1,I2,I3,I4,NULL,NULL,NULL);
COMMENT FIND THE LAST NAME;
    I:=0;  LASTNAM:=BLANK;
    WHILE NAM(I|1)¬=BLANK DO
      BEGIN LASTNAM(I|1):=NAM(I|1);I:=I+1;END;
COMMENT FIND OUT IF THE NAME RING IS EMPTY;
    IF NAMPTR=NULL THEN
      BEGIN
        NAMPTR:=NAMERING(LASTNAM,TDATA,NAMPTR);PTN(NAMPTR):=NAMPTR;
        I1PTR:=I1RING(I1,TDATA,I1PTR);PTB(I1PTR):=I1PTR;
        I4PTR:=I4RING(I4,TDATA,I4PTR);PTE(I4PTR):=I4PTR;
        PN(TDATA):=PTD(NAMPTR);
        PT1(TDATA):=PTA(I1PTR);
        PT4(TDATA):=PTC(I4PTR);
      END
COMMENT CHECK TO SEE IF NAME IS ALREADY PRESENT;
```

```
    ELSE
    BEGIN
      BEGIN
        TEMPN:=NAMPTR;
COMMENT IF THIS IS THE HIGHEST NAME ALPHABETICALLY THEN ADD TO FRONT
OF RING;
          IF LASTNAM<LASTNAME(TEMPN) THEN
            BEGIN
              WHILE (PTN(TEMPN)¬=NAMPTR) DO
                 TEMPN:=PTN(TEMPN);
              NAMPTR:=PTN(TEMPN):=NAMERING(LASTNAM,TDATA,NAMPTR);
              PN(TDATA):=PTD(PTN(TEMPN));
            END
          ELSE
      BEGIN
        WHILE(PTN(TEMPN)¬=NAMPTR)AND(LASTNAME(PTN(TEMPN))<=LASTNAM) DO
               TEMPN:=PTN(TEMPN);
COMMENT IF THE NAME ALREADY EXISTS IN THE RING;
        IF LASTNAM=LASTNAME(TEMPN) THEN
          BEGIN
           TEMP:=PTD(TEMPN);
           IF NAM<NAME(TEMP) THEN
             BEGIN
               WHILE(PN(TEMP)¬=PTD(TEMPN)) DO
                  TEMP:=PN(TEMP);
               PN(TDATA):=PTD(TEMPN);
               PTD(TEMPN):=PN(TEMP):=TDATA;
             END
            ELSE
COMMENT FIND THE PROPER ALPHABETICAL LOCATION FOR THE NEW NAME;
              BEGIN
                WHILE(PN(TEMP)¬=PTD(TEMPN))AND(NAM>=NAME(PN(TEMP)))DO
                      TEMP:=PN(TEMP);
                PN(TDATA):=PN(TEMP);
                PN(TEMP):=TDATA;
              END;
          END
        ELSE
COMMENT SINCE LAST NAME IS NOT EQUAL REARRANGE POINTERS IN NAME RING TO
     ACCOMMODATE NEW RECORD;
          BEGIN
            PTN(TEMPN):=NAMERING(LASTNAM,TDATA,PTN(TEMPN));
            PN(TDATA):=PTD(PTN(TEMPN));
          END;
      END;
      END;
COMMENT NOW CHECK TO SEE IF I1 IS PRESENT;
      BEGIN
```

```
        TEMP1:=I1PTR;
          IF I1<IT1(TEMP1) THEN
            BEGIN
              WHILE (PTB(TEMP1)¬=I1PTR) DO
                 TEMP1:=PTB(TEMP1);
              I1PTR:=PTB(TEMP1):=I1RING(I1,TDATA,I1PTR);
              PT1(TDATA):=PTA(PTB(TEMP1));
            END
          ELSE
      BEGIN
        WHILE(PTB(TEMP1)¬=I1PTR)AND(IT1(PTB(TEMP1))<=I1) DO
              TEMP1:=PTB(TEMP1);
        IF I1=IT1(TEMP1) THEN
          BEGIN
            TEMP:=PTA(TEMP1);
            IF NAM<NAME(TEMP) THEN
              BEGIN
                WHILE(PT1(TEMP)¬=PTA(TEMP1))DO
                  TEMP:=PT1(TEMP);
                PT1(TDATA):=PTA(TEMP1);
                PTA(TEMP1):=PT1(TEMP):=TDATA;
              END
            ELSE
              BEGIN
                WHILE(PT1(TEMP)¬=PTA(TEMP1))AND(NAM>=NAME(PT1(TEMP))) DO
                   TEMP:=PT1(TEMP);
                PT1(TDATA):=PT1(TEMP);
                PT1(TEMP):=TDATA;
              END;
          END
        ELSE
          BEGIN
            PTB(TEMP1):=I1RING(I1,TDATA,PTB(TEMP1));
            PT1(TDATA):=PTA(PTB(TEMP1));
          END;
      END;
      END;
COMMENT NOW CHECK TO SEE IF I4 IS PRESENT;
      BEGIN
        TEMP4:=I4PTR;
          IF I4<IT4(TEMP4) THEN
            BEGIN
              WHILE (PTE(TEMP4)¬=I4PTR) DO
                 TEMP4:=PTE(TEMP4);
              I4PTR:=PTE(TEMP4):=I4RING(I4,TDATA,I4PTR);
              PT4(TDATA):=PTC(PTE(TEMP4));
            END
          ELSE
```

```
      BEGIN
        WHILE(PTE(TEMP4)¬=I4PTR)AND(IT4(PTE(TEMP4))<=I4) DO
            TEMP4:=PTE(TEMP4);
        IF I4=IT4(TEMP4) THEN
          BEGIN
            TEMP:=PTC(TEMP4);
            IF NAM<NAME(TEMP) THEN
              BEGIN
                WHILE(PT4(TEMP)¬=PTC(TEMP4))DO
                  TEMP:=PT4(TEMP);
                PT4(TDATA):=PTC(TEMP4);
                PTC(TEMP4):=PT4(TEMP):=TDATA;
              END
            ELSE
              BEGIN
                WHILE(PT4(TEMP)¬=PTC(TEMP4))AND(NAM>=NAME(PT4(TEMP))) DO
                   TEMP:=PT4(TEMP);
                PT4(TDATA):=PT4(TEMP);
                PT4(TEMP):=TDATA;
              END;
          END
        ELSE
          BEGIN
            PTE(TEMP4):=I4RING(I4,TDATA,PTE(TEMP4));
            PT4(TDATA):=PTC(PTE(TEMP4));
          END;
      END;
      END;
    END;
  END;
END.
```

RING ORGANIZATION SEARCH PROGRAM

```
BEGIN INTEGER II,IJ,IK,NASEARCH,I1SEARCH,I4SEARCH,COMPARE;
    NASEARCH:=575;    I1SEARCH:=0;    I4SEARCH:=0;
              TEMPN:=NAMPTR;
              TEMP1:=I1PTR;
              TEMP4:=I4PTR;
    II:=IJ:=IK:=COMPARE:=0;
    WHILE(II<NASEARCH)OR(IJ<I1SEARCH)OR(IK<I4SEARCH)DO
      BEGIN STRING(3)CHKSTR;
          READCARD(CARD);
          CHKSTR:=CARD(0|3);
                 END;
          IF CHKSTR<"100" THEN
            BEGIN
            IF II<NASEARCH THEN
            BEGIN
            II:=II+1;
              L:=0; LN:=BLANK;
              WHILE CARD(L|1)¬=BLANK DO
               BEGIN LN(L|1):=CARD(L|1);L:=L+1;END;
              WHILE(PTN(TEMPN)¬=NAMPTR)AND(LASTNAME(TEMPN)¬=LN)DO
       BEGIN TEMPN:=PTN(TEMPN);COMPARE:=COMPARE+2;END;
              COMPARE:=COMPARE+2;
            IF(PTN(TEMPN)=NAMPTR)AND(LASTNAME(TEMPN)¬=LN)THEN
             BEGIN WRITE("ERROR ",CARD); GO TO EXIT; END;
            IF (LASTNAME(TEMPN)=LN) THEN
             BEGIN
              TEMP:=PTD(TEMPN);
              WHILE(PN(TEMP)¬=PTD(TEMPN))DO
               BEGIN TEMP:=PN(TEMP);COMPARE:=COMPARE+1;END;
               COMPARE:=COMPARE+1;
             END;
            END;
            END
            ELSE
          BEGIN
          IF CHKSTR<"700" THEN
            BEGIN
            IF IJ<I1SEARCH THEN
            BEGIN
            IJ:=IJ+1;
              WHILE(PTB(TEMP1)¬=I1PTR)AND(IT1(TEMP1)¬=CHKSTR)DO
       BEGIN TEMP1:=PTB(TEMP1);COMPARE:=COMPARE+2;END;
              COMPARE:=COMPARE+2;
            IF(PTB(TEMP1)=I1PTR)AND(IT1(TEMP1)¬=CHKSTR)THEN
```

```
               BEGIN WRITE("ERROR ",CARD); GO TO EXIT; END;
              IF (IT1(TEMP1)=CHKSTR) THEN
               BEGIN
                TEMP:=PTA(TEMP1);
                WHILE(PT1(TEMP)¬=PTA(TEMP1))DO
                 BEGIN TEMP:=PT1(TEMP);COMPARE:=COMPARE+1;END;
                 COMPARE:=COMPARE+1;
               END;
              END;
              END
              ELSE
              BEGIN
              IF IK<I4SEARCH THEN
              BEGIN
              IK:=IK+1;
                WHILE(PTE(TEMP4)¬=I4PTR)AND(IT4(TEMP4)¬=CHKSTR)DO
         BEGIN TEMP4:=PTE(TEMP4);COMPARE:=COMPARE+2;END;
                COMPARE:=COMPARE+2;
              IF(PTE(TEMP4)=I4PTR)AND(IT4(TEMP4)¬=CHKSTR)THEN
               BEGIN WRITE("ERROR ",CARD); GO TO EXIT; END;
              IF (IT4(TEMP4)=CHKSTR) THEN
               BEGIN
                TEMP:=PTC(TEMP4);
                WHILE(PT4(TEMP)¬=PTC(TEMP4))DO
                 BEGIN TEMP:=PT4(TEMP);COMPARE:=COMPARE+1;END;
                 COMPARE:=COMPARE+1;
               END;
              END;
              END;
            END;
 EXIT:     END;
WRITE(" ");
WRITE(" ");
WRITE("LOGICAL COMPARISONS=",COMPARE);
  END;
```

BIBLIOGRAPHY

1. Headquarters United States Marine Corps LTR A03F4-WGH, Promulgation of General Operational Requirement No. CC-9 Marine Corps Tactical Command and Control System (MTACCS), 28 July 1967.

2. Hughes Aircraft Company, MIFASS Technical System Model, V. 11, p. 3-8, 2 March 1970.

3. Hughes Aircraft Company, op CIT, p. 4-6.

4. Interviews with Director MTACCS Test Bed, Camp Pendleton, CA. 27 January and 8 May 1972.

5. HSIAO, David and Harary, Frank, "A Formal System for Information Retrieval from Files," Communications of the ACM, V. 13, p. 67-73, Feb. 1970.

6. DODD G.C., "Elements of Data Management Systems", Computing Surveys, V. 1, p. 120, 2 June 1969.

7. IBID, p. 120-121.

8. Buchholz, W., "File Organization and Addressing", IBM Systems Journal, V. 2, p. 86-111, June 1963.

9. CODASYL, Survey of Generalized Data Base Management Systems, 1969.

10. CODASYL, DATA Base Task Group, 1969.

11. Control Data Corporation, Computer Systems Mars-III/Master Ref. Manual, 1970.

12. Flores, I., Data Structure and Management, Prentice Hall, 1970.

13. IBM, IBM System/360 Model 67 Functional Characteristics, 1967.

14. Johnson, Lyle E., System Structure in Data Programs and Computers, Prentice-Hall, Inc., 1970.

15. Knuth, D. E., Fundamental Algorithms, p. 355, Addison-Wesley, 1969

16. Lum, V. Y., Yuen, P. S. T., Dodd, M., "Key-to-Address Transform Techniques: A Fundamental Performance Study on Large Existing Formatted Files", Communications of the ACM, V. 14 p. 229-231, April 1971.

17. Price, C. E., "Table Lookup Techniques", Computing Surveys, V. 3, p. 61, 2 June 1971.

18. Gries, D., Compiler Construction for Digital Computers, p. 216, Wiley, 1971.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	Assoc Professor U. R. Kodres, Code 53Kr (thesis advisor) Department of Mathematics Naval Postgraduate School Monterey, California 93940	1
4.	LT COL Barry N. Bittner, USMC 459648275 Third Marine Division FPO San Francisco, California 96602	1
5.	MAJ Charles L. Carpenter, Jr., USMC 183284315 Headquarters Battalion Headquarters, U. S. Marine Corps Washington, D. C. 20380	1

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) Naval Postgraduate School Monterey, California 93940	2a. REPORT SECURITY CLASSIFICATION Unclassified 2b. GROUP

3. REPORT TITLE

A Comparative Analysis of File Organizations

4. DESCRIPTIVE NOTES (Type of report and, inclusive dates)

5. AUTHOR(S) (First name, middle initial, last name)

Barry Nicholas Bittner and Charles Lorain Carpenter, Jr.

6. REPORT DATE	7a. TOTAL NO. OF PAGES 82	7b. NO. OF REFS 18

8a. CONTRACT OR GRANT NO. b. PROJECT NO. c. d.	9a. ORIGINATOR'S REPORT NUMBER(S) 9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)

10. DISTRIBUTION STATEMENT

Approved for public release; distribution unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY Naval Postgraduate School Monterey, California 93940

13. ABSTRACT

Increasingly more sophisticated weaponry necessitates that U. S. military organizations insure timely and responsive tactical command and control systems. Automation is one obvious answer towards accomplishing this goal. This paper may be viewed as a simulation study of file organizations which are typical to command and control systems. It reports the findings of a comparative analysis of five different file organizations to determine their responsiveness to five types of commonly used application subroutines. It also uncovers areas for future research with respect to command and control systems' file organizations.

DD FORM 1 NOV 65 1473 (PAGE 1)

S/N 0101-807-6811

Security Classification

A-31408

Security Classification

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Command and Control						
File Organizations						
Generalized File Structure						
Sequential						
Multilist						
Partially Inverted						
Random						
Ring						

DD FORM 1 NOV 65 1473 (BACK)
/N 0101-807-6821

Security Classification

A-31409